# NAVAL POSTGRADUATE SCHOOL
Monterey, California

# THESIS

A REAL-TIME EXECUTIVE FOR
MULTIPLE-COMPUTER CLUSTERS

by

David J. Brewer
December 1984

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution is unlimited

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

| | | |
|---|---|---|
| **REPORT DOCUMENTATION PAGE** | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (*and Subtitle*) A Real-Time Executive for Multiple-Computer Clusters | | 5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December 1984 |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(*s*) David J. Brewer | | 8. CONTRACT OR GRANT NUMBER(*s*) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943 | | 12. REPORT DATE December 1984 |
| | | 13. NUMBER OF PAGES 226 |
| 14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*) Naval Postgraduate School Monterey, California 93943 | | 15. SECURITY CLASS. (*of this report*) Unclassified |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

Distributed Processing, MÚLTIBUS, CP/M-86, Ethernet, Local Area Network

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

This thesis extends the multi-computer real-time executive, MCORTEX, for a cluster of single board computers (INTEL iSBC 86/12 86/12) on the MULTIBUS, to a multiple cluster system tied together by a Local Area Network (Ethernet). The E-MCORTEX system uses eventcounts and sequencers to synchronize processes resident in the network. Data communications between processes are presently limited to a single cluster with shared memory.

(CONTINUED)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

ABSTRACT (Continued)

However, future versions of E-MCORTEX will permit network-wide process synchronization and data communication.

S/N 0102- LF- 014- 6601

# A Real-Time Executive for Multiple-Computer Clusters

by

David J. Brewer
Lieutenant, United States Navy
B.S., University of Idaho, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December, 1984

## ABSTRACT

This thesis extends the multi-computer real-time executive, MCORTEX, for a cluster of single board computers (INTEL iSBC 86/12) on the MULTIBUS, to a multiple cluster system tied together by a Local Area Network (Ethernet). The E-MCORTEX system uses eventcounts and sequencers to synchronize processes resident in the network. Data communications between processes are presently limited to a single cluster with shared memory. However, future versions of E-MCORTEX will permit network-wide process synchronization and data communication.

DISCLAIMER

Some terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurrence of a trademark, all trademarks appearing in this thesis will be listed below, following the firm holding the trademark:

1. INTEL Corporation, Santa Clara, California

| | |
|---|---|
| INTEL | MULTIBUS |
| iSBC 86/12A | INTELLEC MDS |
| ISIS-II | PL/M-86 |
| 8086 | |

2. Digital Research, Pacific Grove, California

| | |
|---|---|
| CP/M-86 | LINK86 |
| PL/I-86 | ASM86 |
| DDT86 | |

3. XEROX Corporation, Stamford, Connecticut

   Ethernet Local Area Network

4. InterLAN Corporation, Westford, Massachusetts

   NI3010 Ethernet Communication Controller Board

## TABLE OF CONTENTS

LIST OF FIGURES

## LIST OF TABLES

# I. INTRODUCTION

## A. DISCUSSION

### 1. General

The purpose of this thesis is to extend the existing version of MCORTEX to a distributed multi-computer real-time executive which transcends the boundaries of a length-limited parallel system's bus, the MULTIBUS. This extension is provided by a local area network (LAN) medium, the Ethernet, and the additional operating system primitives.

As the anti-air warfare (AAW) system of the 1980's for the U.S. Navy, the AEGIS Weapon System captured the attention of a project group at the Naval Postgraduate School (NPS). The project group was formed to look at the AN/SPY-1A phased array radar processing unit. This unit was selected due to the time critical nature of the processing requirements, i.e., the fast reaction to inbound hostile air contacts (missile and aircraft). The AEGIS Modeling Group has been working on the VLSI architecture and the MCORTEX real-time executive for several years.

The fundamental objective is to utilize commercially available LSI and VLSI components that can be implemented in a modular form within the AEGIS Weapons System. Subsequent low cost is a desirable effect, but the proposed replacement of the current four-bay AN/UYK-7 computers, of the AN/SPY-1A phased array suite, is not soley cost-based. Reliability and functional redundancy in the event of failure are extremely important criteria. Mean time to repair (MTTR) is a crucial issue for deployed units (ship or aircraft), due primarily to the unavoidable disrupted Sea Lanes of Communication (SLOC). An onboard technician could discard a failed component and replace a low cost LSI device, such as a microprocessor, from an onboard supply bin.

The project team has produced (up to and including this thesis) a highly modular hardware base, integrated with an equally modular and highly extensible software base. The use of Ethernet as the highest level bus has introduced another commercial-grade product into an existing system of commercial-grade products. As an established standard in the marketplace, the low cost, availability, and support of Ethernet is virtually guaranteed for years to come.

2. Specific

We define a cluster as a group of single board computers (SBC), controlled by multiple kernels of MCORTEX, sharing a common backplane. The integration of the kernels with a general purpose commercially available operating system (CP/M-86) collectively provides multiprogramming capability, multiprocessing capability, and standard disk operating system (DOS) functions. Increased cost-performance is realized by expanding MCORTEX to allow multiple kernels to schedule processes that synchronize and communicate via an intercluster bus (Ethernet). The benefits of interconnecting processing nodes to facilitate information exchange and resource sharing has long been recognized. Those recognized benefits are being applied in the development of extended MCORTEX. The collection of available clusters and the high speed interconnect is collectively known as RTC* (Real-time Cluster Star). As will be seen, the Kleene closure connotes the true power and extensibility of MCORTEX.

The locality of processing modules in a real-time environment is tantamount to speed and efficiency. By effectively co-locating real-time sensors and related processing modules, real-time data aquisition and processing is assured. The use of the Ethernet medium allows the extension of needed process synchronization and interprocess communications to processing nodes which cannot be located physically close enough for shared memory.

As a fully distributed real-time executive, MCORTEX consists of single board resident kernels which support multiprocessing. Process synchronization between virtual processes in the same cluster or in different clusters is provided, entirely transparent to user processes, through integrated cluster hardware and kernel primitives.

The distinction must be made between **user** processes and **system** processes. MCORTEX is the executive which provides primitives to allow processes to synchronize and communicate asynchronously. The only **system** process invoked by MCORTEX is the device-dependent Ethernet Communication Controller Board (ECCB) handler and packet interpreter. This **system** process is resident within one SBC at each cluster. As a consumer of Ethernet Request Packets (ERP), produced by each kernel, this virtual processor does not compete against other processes for a time quantum. It is through the ERP's that user processes make known their need to transmit information over Ethernet. It is transparent to the user processes, however, that an ERP is generated; MCORTEX takes care of this detail. The ECCB handler and packet interpreter is scheduled under MCORTEX and never surrenders the CPU. When it does not have any Ethernet Request Packets to consume, it idles in a "Busy Wait" loop. It is anticipated that its wait will be minimal. **User** processes are those which are independent of cluster hardware, generally cyclic in nature, and provide a function in support of the Aegis Weapon System Simulation and Modeling effort.

## B. BACKGROUND

The initial design of MCORTEX was completed in 1980. The implementation for the iSBC 86/12 single board processors was completed in three Naval Postgraduate School theses in 1981 and 1982. Wasson [Ref. 1] defined the detailed design of an operating system tailored to real-time image processing. His design used the MULTICS concept of segmentation and per process stacks and Reed and Kanodia's [Ref. 2] eventcount synchronization methods. Rapantzikos [Ref. 3] began the initial implementation of Wasson's efforts. At this point, MCORTEX used the concept of a "two level traffic controller" to effect processor multiplexing among eligible processes.

Cox [Ref. 4] simplified the design of MCORTEX. He reduced the traffic controller to one level of abstraction, favoring reduced MCORTEX execution overhead over the security of the two level traffic controller. Cox's other contribution was the addition of a "gatekeeper" module to the entry to the operating system, so the user's access to system calls was simplified. Klinefelter [Ref. 5] generalized Cox's work and developed a technique to dynamically interact with the operating system during its execution.

During the early stages of development of MCORTEX concurrent research efforts, within the AEGIS Modeling Group, were producing a multi-user CP/M-86 based disk sharing environment. It was envisioned this system would be used to develop software in support of the SPY-1A processing emulation.

Rowe [Ref. 6] brought the powerful, highly portable functions of the multi-user CP/M-86 operating system under the control of MCORTEX. He also developed access mechanisms to the MCORTEX supervisor compatible with Digital Research's PL/I-86 language system. User programs could then be developed in a high level, portable language. The kernels of MCORTEX, system processes, and user processes could then be loaded into single board processors from the CP/M-86 environment. Just as importantly, access to the disk sharing capabiltities of the multi-user CP/M-86 system, via MCORTEX processes, was made possible. Rowe's efforts were a culmination of the planned synergism of the individual research projects.

## C. STRUCTURE OF THE THESIS

The goals of this thesis are to:

1. Extend the existing MCORTEX real-time executive for a single cluster of single board computers with shared memory to a real-time executive for a multiple cluster system without shared memory.
2. Extend the existing MCORTEX without introducing substantial changes either to the MCORTEX executive or its primitives.
3. Use the Ethernet interface between clusters to communicate systems data.

Chapter I discusses the overall intent of the AEGIS Weapons System Simulation Project and the emphasis area this thesis covers in accomplishing project goals.

Chapter II presents design concepts and criteria for the original MCORTEX model and the distribution model upon which the extension to MCORTEX is based.

Chapter III is a presentation of the system architecture, with primary emphasis on hardware components.

Chapter IV details the system design of MCORTEX, including the method by which user processes gain access to Ethernet services.

Chapter V is a thorough presentation of the development of user processes and the modifications to the MCORTEX loader.

Chapter VI is a summary of the current state of the system, with particular emphasis on future enhancements and scheduled modifications.

# II. THE EVENTCOUNT MODEL

## A. A MODEL OF SYNCHRONIZATION

A computer system that manages resources used by concurrently operating, independent users requires a mechanism that allows processes to synchronize the use of shared resources.

The most common existing models of synchronization are based upon the principle of **mutual exclusion** and shared data to achieve synchronization. Semaphores [Ref. 7] and monitors [Ref. 8] are based on the concept of mutual exclusion. In this context mutual exclusion is a mechanism that forces the time ordering of execution of pieces of code, called critical sections.

The characteristics of the semaphore and monitor synchronization models have undesirable effects. These effects include complex proofs for program correctness and limitations on applicability to distributed systems.

The model upon which MCORTEX is based is an event oriented model of synchronization in which processes coordinate their activities by signalling and observing events via synchronization variables, known as "eventcounts" and "sequencers." These synchronization variables are interfaces for all interaction among processes. It is normally unneccessary for a process to know the names or residences of other processes.

This model makes no assumptions about the environmental properties of systems and consequently is directly applicable to distributed systems. A distributed system is defined as a system which, due to the lack of a common memory, requires communication among processes to be via communication channels involving unpredictable time delays.

## B. MODEL VARIABLES

### 1. Eventcounts and Sequencers

Unlike the semaphore model, the MCORTEX model solves the synchronization problem in terms of timing constraints on occurrences of events, instead of mutual exclusion. Events are divided into event classes and events of a given class are represented by an associated synchronization variable of the type 'eventcount.' Primitive operations exist that permit processes to signal and observe occurrences of events.

The eventcount alone is inadequate in certain types of timing constraints problems. This type of synchronization problem has the characteristic that the order of different activities is not specified in advance. Instead the synchronization system dynamically defines a total order among them. To deal with this type of constraint a synchronization variable, known as a "sequencer," is needed.

An eventcount is primarily a count of the number of events of a particular class that have occurred in the past. It can be considered a non-negative integer variable whose value never decreases. This is reasonable, since events cannot 'unhappen.'

### 2. Model Primitives

To signal the occurrence of events, an **advance** primitive is used. Two primitives, **await** and **read**, are used to obtain values of eventcounts. A primitive operation **advance**(E) signals the occurrence of an event in the class associated with the eventcount E. This operation increases the integer value of E by 1. The value of the eventcount equals the number of **advance** operations performed on it. The initial value of an eventcount is zero.

A process can observe the value of an eventcount in one of two ways. The value may be read directly using the primitive **read**(E), or the process can block itself until the eventcount reaches a specific value **v** using the **await**(E,v) primitive. The value returned by **read**(E) counts all of the **advance** operations that precede the execution, and may or may not count those in progress during the **read**. The result of **read**(E) is, therefore, a lower bound on the current value of E after the **read**, and an upper bound on the value of E before the **read**.

Frequently a process may not wish to continue executing unless an event, in a class in which it has interest, has occurred. A 'busy wait' could be implemented easily by looping around an execution of a **read**(E) primitive until a specified value of the eventcount, E, is reached. The implication of wasted CPU cycle time is evident and in many instances could be avoided. A process can voluntarily block itself with an **await**(E, v) primitive call. The calling process will remain suspended (i.e., not ready for execution) until the value of E is at least v. Processes written in PL/I - like pseudo-code, as illustrated in Figure 1, demonstrates the use of the **advance** and **await** primitives. The producer and consumer process must synchronize their use of a shared N-cell circular buffer. The circular buffer is implemented as an array in shared memory with indices from Ø to N-1. Two eventcounts MESSAGE_IN and MESSAGE_OUT are used to synchronize the producer and consumer. The producer generates a series of messages by calls on a function "receive_message" and stores the i-th iteration in message_buffer((i-1) mod N). The consumer reads these values out of the buffer in order and consumes them by calling a "xmit_message" subroutine and advancing eventcount MESSAGE_OUT.

The two eventcounts, MESSAGE_IN and MESSAGE_OUT, coordinate the use of the buffer so that:

(1) the consumer does not read the i-th message from the buffer until it has been stored by the producer, and

```
producer: procedure;

     i = 0;
     do while (FOREVER);
       j = read(MESSAGE_IN);
       k = read(MESSAGE_OUT);
       if ((j - k) >= N) then
         call await(MESSAGE_OUT, k + 1);
          /* if difference in eventcount values
             exceeds buffer length then block */
       message_buffer(i MOD N) = receive_message;
       call advance(MESSAGE_IN);
       i = i + 1;
     end; /* do while */

end; /* procedure */




consumer: procedure;

     i = 1;
     do while (FOREVER);
       call await(MESSAGE_IN, i);
         /* if MESSAGE_IN < i then block */
       call xmit_message(message_buffer((i-1) MOD N));
       call advance(MESSAGE_OUT);
       i = i + 1;
     end; /* do while */

end;  /* procedure */
```

Figure 1   Producer-Consumer  Process Synchronization

(2) the producer does not store the (i + N)th value into the buffer until the i-th value has been read by the consumer.

It is important to note that in the above producer-consumer example, each eventcount has only one writer. In the usual semaphore solution both processes would modify the same synchronization variable. For example, let P(S) represent the synchronizing primitive where processes wait for S (some resource) to become greater than zero and then subtract 1 from S before proceeding. Further, let V(S) represent the synchronizing primitive where the processes add 1 to S before proceeding. With this type of synchronization the consumption and production of a result or resource requires that all processes read or write S. A reduction in write competition often occurs in eventcount solutions, resulting in simplified correctness proofs and simplifying the synchronization of physically distributed processes.

The power of eventcounts rest in their ability to achieve synchronization through a relative ordering of events, rather than by mutual exclusion. In the previous example, concurrency of execution is guaranteed if the producer starts out several steps ahead of the consumer and the speeds of production and consumption are equal. In that case there does not exist a time when the consumer or producer must wait for the other to complete an operation.

In synchronization problems that require exclusive use of a resource, the eventcount alone is inadequate. Two or more processes desiring to use a shared resource or write to a shared buffer location are natural examples. Another kind of an object, known as a "sequencer" can be used to provide the needed total ordering. A sequencer is considered a natural number generator, i.e, it returns the sequence 0,1,2,..., etc. Only one operation exists on a sequencer - **ticket**. When applied to a sequencer S, **ticket**(S) returns a non-negative integer value as its result. The **ticket** primitive is based on the idea of the first-come first-served principle used in everyday life. A ticket machine in a catalog sales store or shoe store is an example. The ticket machine issues successive integer values on the ticket, and the next customer to be served is based on the number on the ticket. The store clerk can determine the next person to be served by merely adding one to the previously served number. The customers are served in first-come first-served order. If a customer with the next ticket number has walked out of the store when his number is called, he loses his turn and must get another ticket. This service policy is usually implemented in both stores and in computer operations by a watchdog timer.

The use of sequencers implies mutual exclusion not present in eventcounts. In the case of multiple producers, in a producer-consumer relationship, all message deposits must be mutually exclusive, but it is highly undesirable to place an a priori sequence constraint on several producers. Each producer obtains a ticket number from sequencer S for depositing its message in the buffer. Once a process obtains a ticket, it merely waits for the completion of all producers that obtained prior tickets. Each producer executes the code illustrated in Figure 2 and each consumer executes the same code shown in Figure 1.

The producers block in the following circumstances:

(1) Another producer has a lower ticket value and as yet has not deposited his message.

(2) The single consumer is unable to keep up with the messages deposited in the buffer.

## C. A DISTRIBUTED SYNCHRONIZATION MODEL

### 1. Asynchronous Eventcounts

In distributed clusters without shared memory, a change to an eventcount (via **advance**) takes time to propagate down communication lines to other systems. Two major options exist: (1) implement all eventcounts so that a given eventcount exists only in the cluster where it is most frequently accessed. All other clusters which need the value must make remote accesses to the value. (2) distribute the eventcount values, so that each cluster maintains a local copy. The latter option is that selected in this extension of MCORTEX.

```
producer: procedure;

     do while (FOREVER);
       t = ticket (S);
         /* producers synchronize */
       call await(MESSAGE_IN, t);
         /* at this point in execution it´s this
            processes´ turn, but now must
            synchronize with the consumer */
       j = read(MESSAGE_IN);
       k = read(MESSAGE_OUT);
       if ((j - k) >= N) then
         call await(MESSAGE_OUT, k + 1);
            /* if buffer is full then block */
       message_buffer(t MOD N)=receive_message();
       call advance(MESSAGE_IN);
     end; /* do while */

end;  /* procedure */
```

Figure 2    Multiple-Producers/Single-Consumer Relationship

We define a local eventcount to be an eventcount which is not accessed outside the cluster. We define a remote eventcount as one which is accessed in at least two clusters. A producer at a cluster simply advances an eventcount. If the eventcount is a remote eventcount, the operating system generates the necessary commands which advance a local copy as well as the remote copies of this eventcount. The distributivity of the eventcount is entirely transparent to the producer and consumer. Only the operating system knows in which cluster the producers and consumers reside during their lifetime.

By transmitting the eventcount value, the robustness of the system is assured. Even if the transmission message is lost or not properly received, the very next advance and a subsequent successful transmission will bring the remote copy up to its correct value. The non-decreasing nature of the eventcount value accounts for this robustness.

The design modifications to MCORTEX to allow the eventcount values to be distributed are fully presented and discussed in Chapter 4.

# III. SYSTEM ARCHITECTURE

## A. HARDWARE REQUIREMENTS

### 1. System Configuration

A cluster of Real-Time Cluster Star (RTC*), as shown in Figure 3, is based on the INTEL iSBC 86/12A single board computer (SBC) with MULTIBUS serving as the intracluster bus. Figure 4 illustrates two clusters connected by the Ethernet LAN medium, which serves as the intercluster bus.

Although only four SBC's are shown at each cluster, the limitation is entirely dependent on the number of bus masters. A bus master can drive the command and address lines: it can control the bus. Since multiple bus masters exist in this configuration, some means must be available in hardware to arbitrate their simultaneous requests to use the MULTIBUS. A customized random priority bus resolver, designed specifically for this system, serves a maximum of eight bus masters. A bus slave, such as a RAM board, cannot control the bus and does not require arbitration circuitry.

Two shared memory boards also share the MULTIBUS. A 32K RAM extension board is used as shared memory for process synchronization and control under MCORTEX and for CP/M-86 multi-user system control. A 64K RAM extension board provides additional shared memory required for user

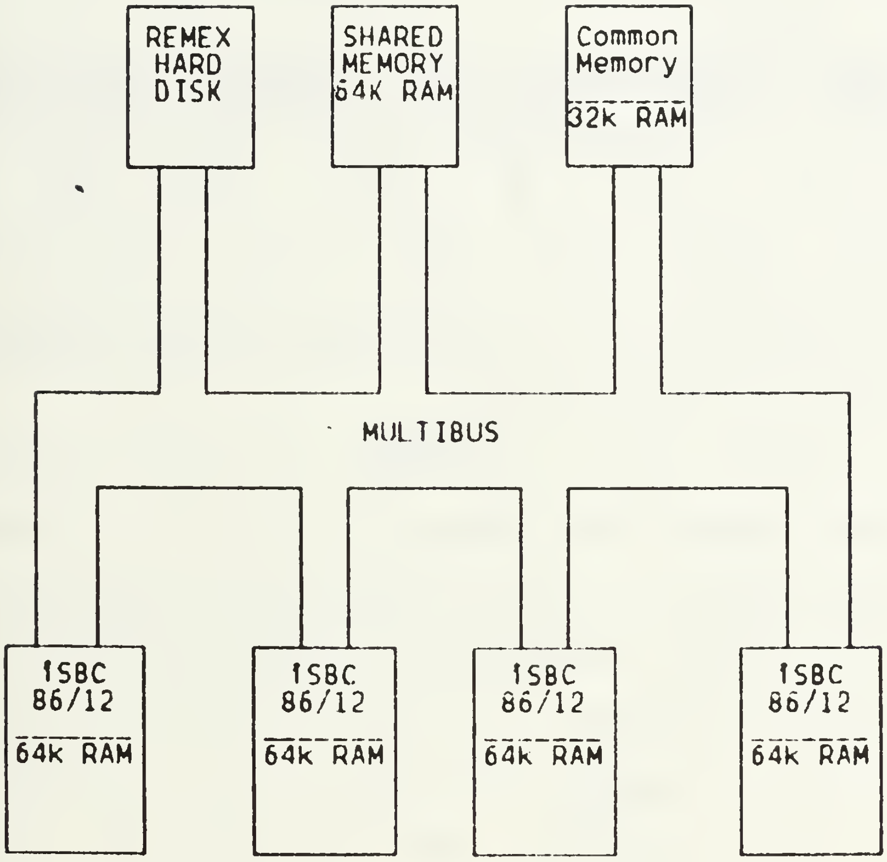


FIGURE 3 Cluster Hardware Configuration

process data communications. Two hard disk systems are available for application process use within a cluster. The REMEX hard disk system has a disk controller card which is placed in an odd slot (required for a bus master) in the MULTIBUS backplane.

The InterLAN NI3010 Ethernet Communications Controller is a MULTIBUS-based single board processor which along with a transceiver provides the cluster with a complete connection to an Ethernet medium. This is the hardware extension to the cluster which allows MCORTEX to be distributed over the Ethernet.

Although only two clusters are shown in Figure 4, the Ethernet specification [Ref. 9] allows for a maximum of 1024 nodes. However, the limiting factor in MCORTEX is the number of clusters that can be addressed with the current packet routing algorithm. As will be discussed in Chapter 4, the upper bound is 16 clusters which is more than adequate considering the current availability of only two NI3010 boards and three NI3210 boards (enhanced version of the NI3010) in the AEGIS Simulation Laboratory at the U.S. Naval Postgraduate School, Monterey, California.

An experimental system that contains both analogous and dissimilar components to that of RTC* is CM* [Ref. 10]. The most important comparison is between CM*'s Kmap and RTC*'s NI3010 and Driver. The Kmap must effectively route a

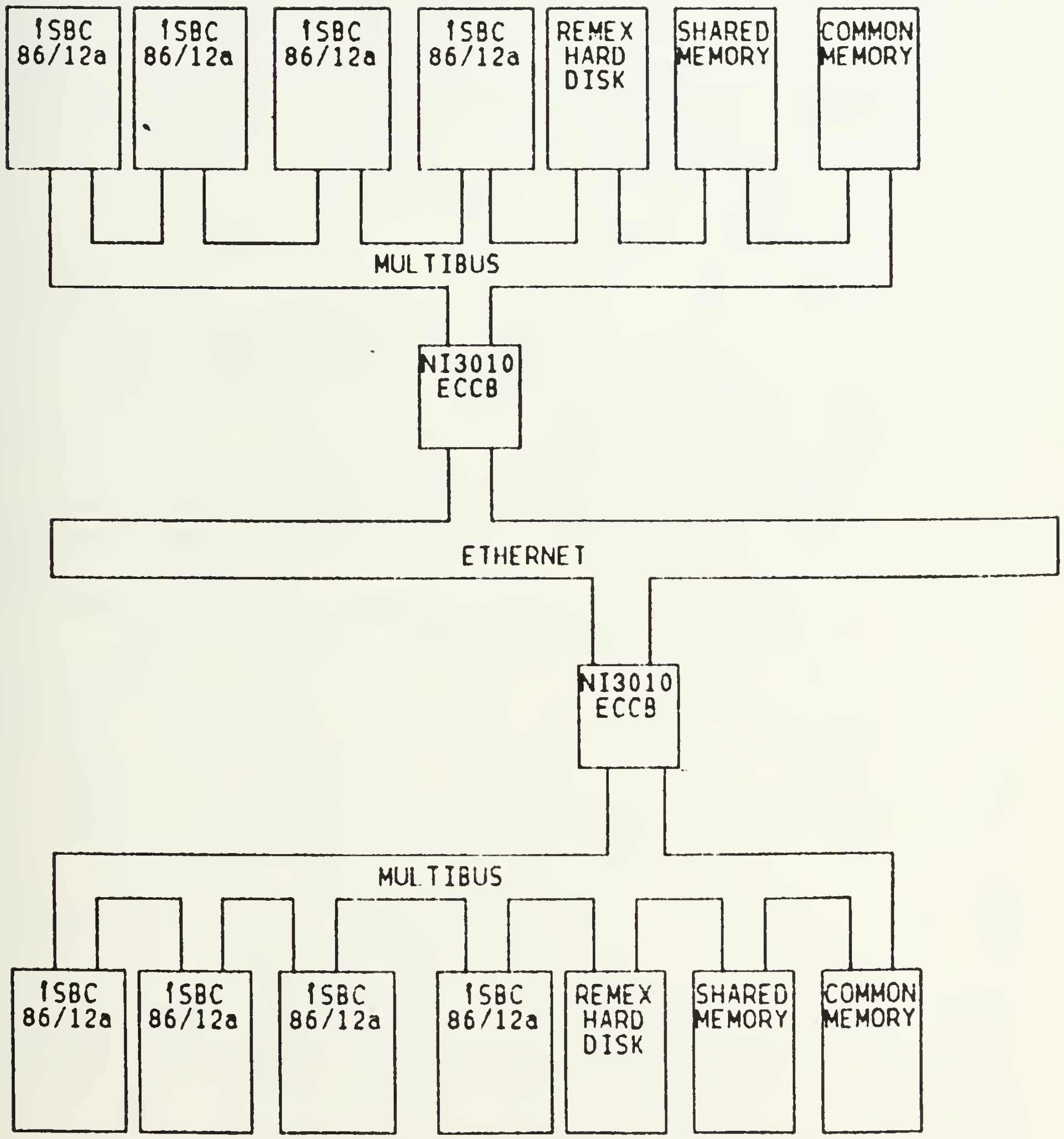


FIGURE 4 Real-Time Cluster STAR Architecture

shared address between clusters, whereas the NI3010 Driver routes an entire datagram of information. The intercluster response time of Kmap is on the order of 36 microseconds, while the Ethernet is on the order of milliseconds. Therefore, the use of Ethernet is appropriate where relatively long messages with not very demanding response times are used. The Kmap has a relatively low transfer rate with fast response times. Additionally, the cost of the NI3010 and the Driver development, the flexibility, and its extensibility is far superior to the Kmap. The NI3210 is expected to further increase the speed and efficiency of intercluster communications.

### 2. The iSBC 86/12A Single Board Computer

The iSBC 86/12A board includes a 16-bit CPU, 64K bytes of dynamic RAM, a serial communications interface, three 8-bit programmable parallel I/O ports, programmable timers, priority interrupt control, MULTIBUS interface control logic, and bus expansion drivers for interface with other MULTIBUS interface-compatible expansion boards. The iSBC 86/12A board has an internal bus for all onboard memory and I/O operations and accesses MULTIBUS for all external memory and I/O operations. Therefore, local (onboard) operations do not disturb the MULTIBUS interface available for parallel processing when several bus masters (e.g., DMA devices and other SBC's) are operating concurrently.

The iSBC86/12A provides a three level hierarchical bus structure. At the first level, the 8086 processor communicates through the on board bus with up to 32K of ROM, with serial and parallel I/O ports and with the dual-port bus. Control and access to local RAM is provided by the second level dual-port bus. The third bus level, the MULTIBUS interface, provides access to the MULTIBUS. The presently used wiring option prohibits off board access to local RAM, so that the local RAM is protected from external contamination.

### 3. The 8Ø86 Microprocessor

The 8Ø86 microprocessor, the heart of the single board computer, performs the system processing functions and generates the address and control signals to access memory and I/O devices.

This high performance, general purpose microprocessor base of the iSBC86/12A contains an Execution Unit (EU) and a Bus Interface Unit (BIU). EU functions are supported by instruction fetches and operand reads and writes conducted by the BIU. The BIU can stack instructions in an internal RAM to a level of six deep increasing EU efficiency and decreasing bus idle time. A 16-bit arithmetic/logic unit (ALU) in the EU maintains the CPU status and control flags, and manipulates the general registers and instruction operands. All registers and data paths in the EU are 16 bits wide for fast internal transfers.

The 8086 has eight 16 bit general purpose registers. Four byte addressable registers, known as the data registers, can be used without constraint in most arithmetic and logic operations. The remaining four are primarily pointer registers, but can be used as accumulators. Additionally, the 8086 has four segment registers, an instruction pointer register and a flag register with nine status bits.

The 8086 can address up to one megabyte of memory, 'viewed' as a group of segments, as defined by the application. A segment is a logical unit of memory that may be up to 64K bytes long. The segment registers point to the four currently addressable segments. Programs obtain access to code and data in other segments by changing the segment registers to point to the desired segments.

It is convenient to think of every memory location as having two kinds of addresses, physical and logical. A physical address is a 20-bit value that uniquely identifies each byte location in the megabyte address space. Physical addresses range from 0H through FFFFFH. Programs, however, deal with logical instead of physical addresses. A logical address consists of a base value and an offset value. Whenever the BIU accesses memory - to fetch an instruction or to obtain or store a variable - it generates a physical address from the logical one. This is accomplished by shifting the base value left four bits and adding the offset. The resultant 20 bit value is then used to access memory.

4. Ethernet

Ethernet is a local area network (LAN) optimized for the high-speed exchange of data between information processing equipment within a moderate-sized geographic area. It is the result of a collaborative effort by Digital Equipment Corporation, Xerox Corporation, and Intel Corporation. The Ethernet specification [Ref. 9] provides precise, detailed design information for a baseband local area network and, for brevity's sake, only general aspects pertaining to the RTC* implementation will be discussed here.

Ethernet implements the lowest two layers of the 7-layer OSI/ISO model [Ref. 11 pp. 46-53]. The Data Link layer defines the format and addressing of packets that are broadcast over the "Ether", detects transmission errors, controls access of the network by nodes, and allocates channel capacity. These functions are, in fact, implemented in the NI3010 Ethernet to MULTIBUS communications controller board. The functions carried out by this layer for sending and receiving transmissions are as follows.

a. **Data Encapsulation/Decapsulation**

Defining the format of message packets - the different fields of information within the packets.

Constructing packets from data supplied by the nodes through the higher layers; disassembling network messages and supplying data to the higher layer protocols of the node.

Addressing - handling of source and destination addresses.

Error detection - physical channel transmission errors.

b. **Link Management**

Channel allocation - the length of time of channel use is determined by the packet size.

Channel access - access to the channel is controlled by a contention-avoidance-and-resolution technique, called CSMA/CD, part of which is carried out in each of the two layers. The Data Link level responds to the channel or carrier sensing of the Physical layer. This means that the sender defers sending in the case of traffic, sends in the absence of traffic, and backs off and resends the message a random time interval later in the case of collisions.

The construction and processing of the packets that are transmitted on the Ethernet, is part of the data encapsulation function of the Data Link layer. The Ethernet packet is made up of five fields, as shown in Figure 5 (all bytes are eight bits in length). The smallest total size of a packet transmitted over Ethernet is 64 bytes, and the maximum size of a packet is 1,518 bytes (these figures do not include the eight-byte **preamble**). Details of the fields are included in [Ref. 9], so the only field discussed in detail will be the **destination address**. Knowledge of this field will simplify the discussion of the packet routing algorithm presented in Chapter 4.

A packet can be sent to one, several, or all nodes simultaneously, through unique broadcasting and addressing capabilities. The address of the node (or nodes) that the packet is intended for is placed in this field, which is six bytes in length. A node address can be one of two types:

**Physical address** - the unique address of a single node on any Ethernet.

**Multicast address** - a multidestination address of one or more given nodes on a given Ethernet, of which there are two kinds:

**multicast group address** - virtually any number of node groups can be assigned a group address so they are all able to receive the same packet in a single transmission by a sending node. This is a key feature in the packet routing algorithm to be discussed in Chapter 4.

**broadcast address** - a single multicast address by which a packet can be sent to the set of all nodes on a given Ethernet.

The first bit in the Destination Address field is set to indicate a physical or multicast address. The remaining 47 bits specify the address itself. If a packet is to be broadcast to all nodes, the 47 bits are all set to "1." The 47 remaining bits allow for 2 ** 47 (over 147 trillion) possible addresses.

The **Physical Layer** of Ethernet provides a ten-million-bit-per-second channel over a coaxial cable medium. It specifies all the essential physical characteristics of Ethernet, including bit encoding, timing, voltage levels, and two compatibility interfaces.

The main functions of this layer are:

**Data encoding/decoding:**

Generation and removal of 64 preamble bits before each packet is transmitted for synchronization and timing of messages.

Bit encoding and decoding - between the binary encoded form of the Data Link level and the phase encoded form required for transmission on the coaxial cable. **Manchester phase encoding** is specified for all data transmitted on the Ethernet at a data transmission rate of ten million bits per second (10 Mbps).

**Channel Access**

Transmission and reception of encoded data.

Carrier sense - monitoring the channel for traffic and signaling the Data Link layer if traffic is detected.

Collision detect - signaling the Data Link layer, _during_ transmission, when a collision is detected.

Two important compatibility interfaces, the transceiver cable interface and the coaxial cable interface, are also specified in the Physical layer. Detailed information regarding these interfaces is contained in [Ref. 9].

5. _NI3010 Ethernet Communication Controller Board_

In the following discussion of the NI3010's operation, reference to a "host" is synonymous with a single board computer in a cluster which contains the device driver for the NI3010 board. Details concerning this driver's system role are contained in Chapter 4.

The NI3010 ECCB is a MULTIBUS-compatible component that implements layers one and two of the ISO/OSI 7-layer model. Although programmable as a polled or interrupt-driven

DMA device, it is used entirely as an interrupt-driven component in this implementation. The NI3010 serves as a bus master when controlling the DMA operations between the NI3010 buffers and the host's memory, and as a slave to the commands of the host.

The host controls the NI3010 by writing to onboard registers which are MULTIBUS addressable I/O ports. Depending on the state of execution, the host may direct the NI3010:

(1) To perform a load command

(2) In preparation for a DMA operation - load a memory address and a byte count, or

(3) To enable an interrupt register, to inform the host when a directed operation is complete.

The host programs the NI3010 by writing a command to the command register, whose I/O address is currently set at B0H (base register). The command function codes are contained in Table 3-1 of [Ref. 12]. After issuing a command, the host must check for a value in the Command Status Register. The details of this read operation are covered in [Ref. 12], but briefly: Any value other than zero or one in the Command Status Register, following execution, represents a board failure. If at any time during MCORTEX execution a diagnostic appears that indicates an NI3010 board failure, the RTC* system operator can run a diagnostic program that fully exercises the board. The code and invocation procedures for this diagnostic routine is contained in Appendix L.

Of particular importance is the requirement to read the Command Status Register at the beginning of any code that controls the NI3010. This is neccessary because of the power-up diagnostic that runs at system start-up or due to a MULTIBUS reset. This automatic testing feature places a value in the Status Register that must be read to clear the register before any commands can be issued to the NI3010.

The NI3010 transmit process consists of obtaining data packets from shared data memory, via a DMA operation, forming them into Ethernet frames, and successfully delivering them to the intercluster bus (the "Ether").

The following describes what happens when a transmit packet goes from MULTIBUS memory to the NI3010:

(1) The host writes an interrupt code of zero to the interupt enable register on the NI3010. Writing this register clears the NI3010's interrupt line (currently set for interrupt 5).

    **Note:** This step ensures that the DMA controller does not start a DMA transfer as soon as the byte count registers contain a non-zero value.

(2) The host writes a 24-bit MULTIBUS memory address into the NI3010's bus address registers.

(3) The host writes the packet's byte count into the NI3010's byte count registers.

(4) The host initiates a DMA transfer by writing to the interrupt enable register an interrupt code of 6. The NI3010 will now interrupt the host processor when it completes the DMA transfer.

(5) The NI3010 moves the transmit packet from host memory to its transmit buffer (only one packet at a time may be resident in this buffer). After accepting each data byte, the DMA controller increments the address in the bus address registers and decrements the byte count in the byte count registers. When the byte count reaches zero and its transmit register is empty, the NI3010 interrupts the host processor. This is a transmit-DMA-done (TDD) interrupt. The transmit data is now stored in the transmit buffer.

(6) To transmit this data on the Ethernet, the host issues a Load transmit Data and Send command (29H). The NI3010 carries out the command, reflecting its status in the register. The host must read the status register.

The following describes what happens when a receive packet goes from the NI3010's receive queue (16K byte capacity) to MULTIBUS memory:

(1) The host issues an interrupt code of 4. This enables a receive-block-available (RBA) interrupt from the NI3010.

(2) The host gets a receive-block-available interrupt. The host now knows that the NI3010's receive queue has a frame awaiting transfer to MULTIBUS memory.

(3) The host writes an interrupt code of zero to the NI3010's interrupt enable register. Writing this register clears the NI3010's interrupt line.

**Note:** Just as in the transmit process, this step ensures that the DMA controller does not start a DMA transfer as soon as the byte count register contains a non-zero value.

(4) The host writes the 24-bit MULTIBUS memory address into the NI3010's bus address registers.

(5) The host writes the byte count of its MULTIBUS buffer into the NI3010's byte count registers.

(6) The host initiates a DMA transfer. It does this by issuing an interrupt code of 7. This also enables a receive-DMA-done interrupt (RDD) from the NI3010.

(7) The NI3010 moves the received frame from its receive queue to host memory. The NI3010 preceeds the packet with a frame status byte, a null byte, and two bytes containing the frame's byte length. After transferring each data byte, the DMA controller increments the address in the bus address registers and decrements the byte count in the byte count registers. The NI3010 generates a receive-DMA-done interrupt when it finishes transferring the frame or when the byte count reaches zero.

(8) The host responds to the RDD interrupt by issuing an interrupt code of zero, disabling the interrupt from the NI3010 board.

The determination of the order in which commands are given is entirely dependent on the application. The 16K byte receive buffer allows the host to read this buffer (via RDD interrupt operation) at its own convenience. This buffers the MULTIBUS from the unpredictable arrival times of intercluser traffic, consequently reducing the time-critical service requirements on the receiving cluster. In contrast is the 2K byte, single packet, transmit buffer. The host system should strive to favor outbound packets to reduce the processing delay by any processors in the cluster.

## B. SOFTWARE SERVICES

### 1. Operating Systems

A copy of a kernel of MCORTEX resides in each processor's local memory and is a part of the address space of each local process. Additionally, GLOBAL memory is accessible to MCORTEX to facilitate interprocess synchronization. Processes are scheduled for execution by a kernel of MCORTEX on each SBC. Any process that advances an eventcount or calls the await primitive "risks" surrendering the CPU to a higher priority ready to run process. A call to the advance primitive always results in a call to the scheduler. If the calling process is still the highest priority "ready to run" process, it will continue in its execution, otherwise another virtual processor will be scheduled to run and the original process will be blocked ("ready' if an advance operation, 'waiting' if an await operation).

In the event there are no user processes in the ready state, the kernel's idle process will run. This process blocks itself every 4 milliseconds and calls the kernel scheduler. If any offboard operation caused an onboard process to be readied, as the only process "ready to run", it will be scheduled. The idle process is always "ready to run", of course, but it has the lowest possible priority.

This implementation of MCORTEX is a major change in the philosophy of previous versions, whereby a system interrupt under MCORTEX control, in conjunction with interrupt flags maintained in GLOBAL memory, provided communication initiation between real processors. Upon receiving an interrupt, each processor checked its flag in GLOBAL memory to determine if the interrupt was intended for a process in its local memory. If not, the process executing at the time of the interrupt continued. Otherwise a call was made to the MCORTEX scheduler and the highest priority ready process was given control of the CPU. For communication between processes in local memory, no interrupt was issued, a call to the scheduler was made directly.

The use of the interrupt was inconsistent with the philosophy of switching processes only at "safe" points in their execution. These "safe" points were required because of non-reentrant PL/I-86 user process code. An interrupt must not occur during a call to a PL/I procedure that is shared among multiplexed processes. Therefore, the original design had a design error which needed correction.

Also, the use of a preemptive interrupt to signal a possible change to all real processors in a cluster was somewhat counterproductive. To cause all real processors to be disrupted in their execution, just because as few as one virtual processor was made ready, is unjustifiable. However, this preemptive interrupt structure has been maintained in MCORTEX in the event a high priority process must be scheduled. A primitive known as PREEMPT, provides this capability. The PREEMPT primitive is the mechanism to schedule time urgent processing which is vital in real-time systems. PREEMPT, however, must be used carefully and sparingly. Processes that are time critical must only use reentrant code, so that when a return from the time critical process is completed, the state of the system is not disturbed.

Access to MCORTEX is through the supervisor at the outermost layer of the MCORTEX four level structure discussed by Klinefelter [Ref. 5 : pp. 44-46].

Also resident in each local memory, if required, is the CP/M-86 operating system. In this configuration the full range of CP/M-86 utilities, [Ref. 13] and [Ref. 14], is available to the user. Additionally, development of user processes can make use of any of the broad scope of commercially available products compatible with CP/M-86. Figure 5 gives a representation of the locations of the system code. The diagram includes the location of DDT-86 as required for a debugging session. A developer of user processes should anticipate needing this powerful debugging tool; the space should remain reserved. Also depicted are the locations of the MCORTEX/MXTRACE loaders. During load, loader memory is not reserved, and care must be taken to ensure that a CMD module's code or data section does not overwrite it. It is permissible, however, to include this memory as part of a module stack or free space, since these structures are developed at module runtime when loader functions have been completed.

### 2. User Processes

User processes may be located in areas indicated in Figure 6. Additionally, if CP/M-86 utilities are not

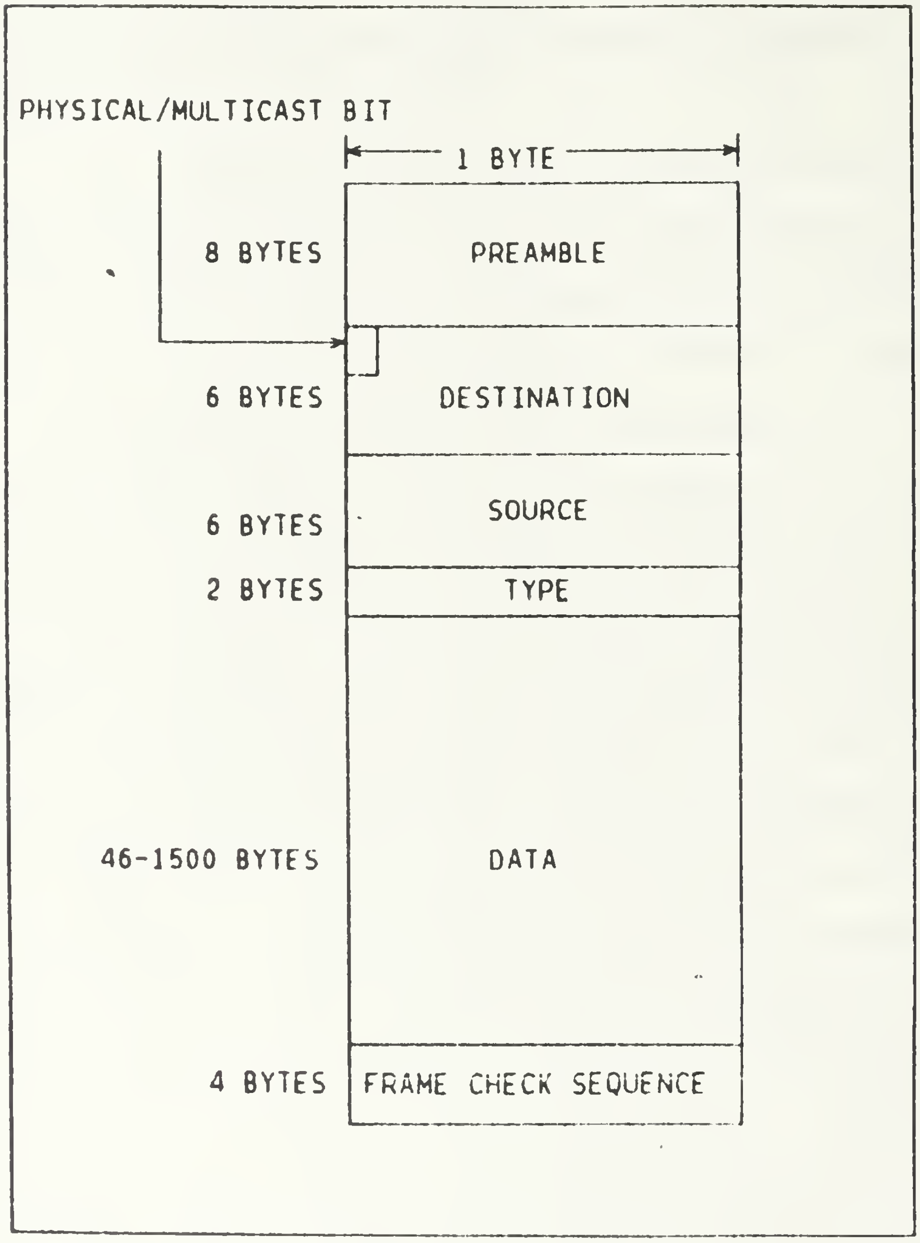
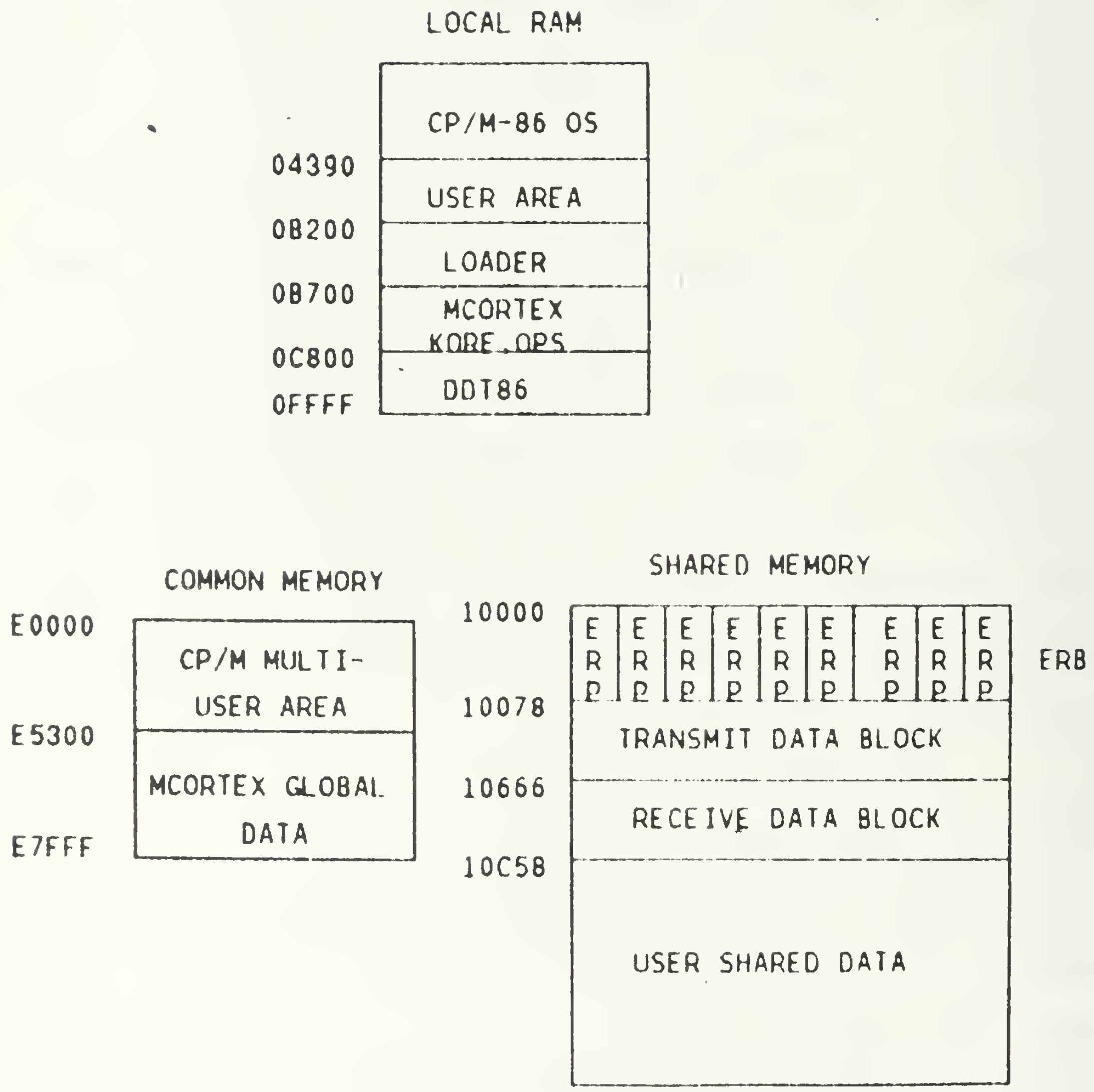


FIGURE 5 Ethernet Packet Format

required, memory reserved for CP/M-86 may hold user processes.

Descriptions of processes in memory are provided to MCORTEX through the CREATE$PROC primitive. This MCORTEX function gives the process a unique identification number, priority, stack (SS and SP registers), next execution address (CS and IP registers), data segment (DS register), and extra segment (ES register). MCORTEX establishes the process initial context using this information to create a virtual processor, which is a software abstraction of a real processor. The virtual processor exists as a combination of data, both in GLOBAL memory, and in each process stack. When executing, the virtual processor becomes identical with the real processor state. Relinquishing the CPU forces the virtual processor status into GLOBAL memory and the process stack into local memory.

As described by Rowe [Ref. 6 : p. 28], special effort has been made to accommodate processes created under PL/I-86 and linked using LINK86. LINK86 concatenates all PL/I-86 code segments into one segment and data segments into one segment. Thus, PL/I-86 processes consist of a series of contiguous code segments followed by a series of contiguous data segments. Additionally, at run time PL/I-86 routines create a stack following the data area. and a free space following the stack. The resulting configuration is shown in Figure 3 of [Ref. 6].



ERB - Ethernet Request Block

ERP - Ethernet Request Packet

FIGURF 6 Cluster Memorv Map

Access to all data areas resulting from a single link, is referenced to a common data segment. Stack pointers are referenced to the stack segment register, and free space pointers to the extra segment register. Additionally, some PL/I-86 runtime routines assume the contents of all three segment registers (DS, SS, ES) are identical.

The MCORTEX CREATE$PROC parameters include the absolute location of process start, stack, and data. For this reason it is advantageous to locate processes absolutely when linking. LINK86 provides such an option [Ref. 13 : p. 7.6], however, the ABSOLUTE option is applicable to the entire CMD file created and cannot be used to distribute the file non-contiguously in memory.

Rowe [Ref. 6] experienced some difficulty using LINK86 as described in [Ref. 13]. His observation was entirely correct, but it was easily corrected by generating a new CP/M-86 operating system using Version 1.1 CCP and BDOS (integrated with a modified BIOS). Version 1.0 contained an error that caused the 128 byte header, preceding CMD files, to be parsed incorrectly at file load time. Details concerning this header are contained in [Ref. 14]. The BIOS was modified due to the removal of the bubble memory board from the multi-user CP/M-86 system. This process of generating a new CP/M-86 operating system is described in adequate detail in [Ref. 14]. The details concerning the multi-user CP/M-86 system BIOS is described in [Ref. 15] and will not be reiterated here.

# IV. DETAILED SYSTEM DESIGN

## A. DESIGN ISSUES

### 1. Real-Time Processing

Real-Time processes are of a time-critical nature, and as such are always resident in memory. The time required to swap a real-time process out of memory, to make room for another, would consume the very same resource being allocated - the CPU. The early designers of MCORTEX considered this issue carefully and the result is an operating system that minimizes context switching overhead. MCORTEX processes reside permanently in memory (once loaded) and only CPU registers, critical to a context switch, are modified. Just as important are issues such as: (1) allocation of shared resources, (2) process integrity, (3) process synchronization, and (4) interprocess communication.

### 2. Shared Resources

Within a cluster (Figure 3) are three critical shared resources : the NI3010 ECCB (i.e., Ethernet), common memory, and shared memory itself. The hierarchical bus structure limits the access of each real processor to common memory and shared memory, and the bus arbiter grants access in a random manner. Each processor executes processes in its own local RAM and only makes memory accesses outside that range when MCORTEX primitives (access to common memory) are used or data computed by a producer must be placed in shared memory for consumption by another process within that cluster. MCORTEX performs its functions by setting up a section of common memory called GLOBAL memory. Table 1 shows how this shared resource is logically organized (Appendix H contains the actual memory locations).

Access to GLOBAL memory is resolved through the combination of a hardware bus lock (LOCK prefix preceding a machine level instruction), and a software lock (GLOBAL$LOCK) located in GLOBAL memory. MCORTEX primitives that access GLOBAL memory set the hardware bus lock through the PL/M-86 function LOCK$SET [Ref. 16]. The real processor executing the kernel, that is executing LOCK$SET , is given sole access to the MULTIBUS for the duration of a single instruction. A LOCK prefix preceding an XCHG instruction causes a value in a register (contents 77H) to be exchanged with GLOBAL$LOCK. The processor then examines the contents of the exchange register. If the register now contains zero, the processor is granted access, if not, the kernel repeats the procedure until a zero is obtained from GLOBAL$LOCK. The XCHG instruction requires two bus cycles to swap 8-bit values, thus without the LOCK prefix it is possible for another processor to obtain the bus between cycles and gain access to the partially-updated GLOBAL$LOCK semaphore. When relinquishing the software lock, the kernel

TABLE 1 - GLOBAL MEMORY

| OFFSET | MNEMONIC | TYPE | INIT | REMARKS |
|---|---|---|---|---|
| 0 | LOCAL$CLUSTER$ADDR | W | X | Address of this cluster |
| | EVC$TBL(100) | S | | Event count table |
| 2 | EVC$NAME | B | FF | Event count name |
| 3 | VALUE | W | 0 | Event count value |
| 5 | REMOTE$ADDR | W | FF | Remote addr of remote copy |
| 7 | THREAD | B | FF | Event count thread |
| | VPM(MAX$CPU * MAX$VPMS$CPU) | S | | Virtual processor map (MAX$CPU = 10, MAX$VPMS$CPU = 10) |
| 602 | VP$ID | B | X | Virtual processor ID |
| 603 | VP$STATE | B | X | Virtual processor state |
| 604 | VP$PRIORITY | B | X | Virtual processor pri. |
| 605 | EVC$AW$VALUE | W | X | Count awaited |
| 607 | SP$REG | W | X | Stack pointer register |
| 609 | SS$REG | W | X | Stack segment register |
| 1602 | GLOBAL$LOCK | B | 0 | |
| 1603 | NR$RPS | B | 0 | # of real processors |
| 1604 | NR$VPS(MAX$CPU) | B | 0 | # of virtual processors (one byte for each possible CPU, MAX$CPU currently = 10) |
| 1614 | HDW$INT$FLAG(MAX$CPU) | B | X | H/W interrupt flag (one for each possible CPU, MAX$CPU currently = 10) |
| 1624 | EVENTS | B | 1 | Number of events |
| 1625 | CPU$INIT | B | 0 | Log in CPU number |
| 1626 | SEQUENCERS | B | 0 | Number of sequencers |
| | SEQ$TABLE(100) | S | | Sequencer table |
| 1627 | SEQ$NAME | B | X | Name of sequencer |
| 1628 | SEQ$VALUE | W | X | Value of sequencer |
| 1927 | | | | |

B - byte W - word S - structure X - not initialized merely sets GLOBAL$LOCK to zero. The "granularity of locking' by the kernels, is all of GLOBAL memory, i.e., no two kernels have access to GLOBAL memory simultaneously.

Users have no access to GLOBAL memory, however MCORTEX provides for user control of shared resources through data held in GLOBAL memory. Sequencers, located in the sequencer table section of GLOBAL memory, are used to provide a turn taking mechanism. Each shared resource is assigned a corresponding sequencer. When processes require a resource, they request a turn through the supervisory function call TICKET, specifying the applicable sequencer. TICKET returns a number indicating the callers turn at the required resource. TICKET advances the sequencer value in GLOBAL memory so that succeeding requests receive higher numbers. Given the situation where a "busy wait" is not to be employed, a process requesting the resource then makes another supervisory call, this time on AWAIT, providing both an identification of the resource and the process turn number. If the resource is not busy, the process will receive immediate access, otherwise the process gives up the CPU.

3. Process Integrity

The design of MCORTEX relies heavily on user cooperation for process integrity. The supervisor controls access to the MCORTEX functions, but even this is a software control and a process that intentionally or inadvertently destroys GLOBAL data would be disastrous. Although local RAM of a processor is inaccessible from MULTIBUS, thus protected from a 'runaway' process, common memory and shared memory are not. Protection from this type of failure requires hardware protection not presently in the system. The low cost of microcomputers however, allows for redundant back up systems which can limit the effects of such failure due to a processor hardware fault.

4. Process Synchronization

Process synchronization is accomplished under MCORTEX through the functions ADVANCE, AWAIT, and PREEMPT. These synchronizing primitives are supported with the functions CREATE$EVC, CREATE$SEQ, READ, DEFINE$CLUSTER, DISTRIBUTION$MAP, and TICKET. Consumer processes use AWAIT to ensure that data they require is ready. Producer processes use ADVANCE to inform consumers that a new iteration of data has been computed. PREEMPT is used by one process to directly ready another process. This primitive is for activation of high priority system processes of a highly time critical nature. A call on a synchronizing primitive may, or may not result in relinquishing the CPU. The CPU is always assigned to the highest priority ready virtual processor on each board regardless of which synchronization function envoked the scheduler (except for PREEMPT, of course).

Before using ADVANCE or AWAIT, an eventcount must be created using CREATE$EVC. Consumers and producers then communicate using the agreed upon eventcount. The current value of an eventcount can be determined through a call on READ. The functions of CREATE$SEQ and TICKET are as discussed earlier, but with broader applications.

The only entity presently distributed by MCORTEX over Ethernet is eventcounts. However, this feature alone allows distributed processes to synchronize. The manner in which processes synchronize is no different than that already discussed. The fundamental issue then becomes the means by which an eventcount of interest can be made available to a producing or consuming process.

Eventcounts may be used in any number of combinations. Producing and consuming processes may be resident in the same cluster, different clusters, or mixed (i.e., a producer and one consumer in the same cluster, with another consumer of the same data type in another cluster). Processes are not aware, however, as to their own distribution - they continue to advance eventcounts and await values just as they always did. This transparency is provided through the primitives DEFINE$CLUSTER and DISTRIBUTION$MAP.

DEFINE$CLUSTER is a procedure that assigns a 16-bit address (the last two bytes of the destination field of an Ethernet packet) to a cluster, and DISTRIBUTION$MAP causes the "remote$addr" field of an eventcount name (see Table 1) to be assigned a value. It is necessary to statically manage the distribution of eventcounts, just as it is necessary to statically manage blocks of shared memory for user processes. It is a decision that must be made by personnel responsible for the development of AEGIS software that will run on PTC* under MCORTEX.

A user process does not need to know the address of the cluster in which it resides, nor is it required to know the cluster addresses of processes that it synchronizes with. Therefore, DEFINE$CLUSTER and DISTRIBUTION$MAP are _not_ primitives called by a user process, but by a system process that calls these primitives in its initialization module. As mentioned before, eventcounts must be created prior to their use. The convention of MCORTEX is that user processes do not create or define them (as a constant) in any way. The same system process that calls DEFINE$CLUSTER and DISTRIBUTION$MAP, also creates all user and system eventcounts and sequencers. Thus, symbolic names only are used by user processes at run-time and the system initialization module at creation time, providing a level of security. It will be seen later how this security is even further enhanced. The manner in which user and system processes are created is covered in complete detail in Chapter V.

5. Interprocess Communication

MCORTEX, at this stage of development, does not provide any means by which data (produced) can be transmitted between clusters. Within the same cluster, however, shared data is stored for consumption in the 64K byte RAM shared memory board. Any buffering of data by user processes must be done explicitly. There is no dynamic allocation of this resource.

With Ethernet serving as the intercluster bus, with eventual data transfer planned, due consideration must be given to the distribution of user processes within RTC*. Processes with a high interprocess communication rate should be located as close together as possible. When this is not feasible, a fairly high efficiency penalty will have to be paid. The Ethernet is clearly the highest level bus in RTC* and memory located at a remote cluster must be viewed as the highest level memory in the memory hierarchy of RTC*. As such, a nonlocal memory access should be avoided as much as possible, but it will never be entirely avoidable. Clearly average memory access times will drop as the rate of local memory references increase. In a distributed system such as RTC*, the nonlocal "hits" on memory should be kept to a minumum. To reiterate, if high volume communicating processes can possibly reside in the same cluster, then they should be so located.

B. ETHERNET ACCESS

1. Cluster Input/Output

MCORTEX must provide a means to transmit copies of values of eventcounts to a remote cluster. This operation must be entirely transparent to user processes, since they have no knowledge of their distributivity.

Figure 7 illustrates an abstraction of the flow of data and control signals necessary to achieve a transmission over Ethernet. It embodies the principles of a flow chart, as well as an abstraction of processing modules and control signals. Refer to Figure 7 for the following discussion. The user processes resident in either SBC 1 or 2 advance an eventcount through the **ADVANCE** primitive operation. The **ADVANCE** primitive makes a determination as to the locality of the eventcount and calls the internal routine SYSTEM$IO only if the eventcount is distributed, i.e., a remote copy is needed at another cluster. The SYSTEM$IO routine makes a determination as to the eventcount communication path (currently the only option is Ethernet). Since the path is Ethernet, the SYSTEM$IO routine writes an Ethernet Request Packet (ERP) to a circular buffer in shared memory, known as the Ethernet Request Block (ERB).

As a shared resource among MCORTEX kernels, an ERP slot in the ERB must be arbitrated for. The TICKET mechanism is employed in SYSTEM$IO, and the circular buffer (ERB) contains ERP's that must be processed. The SYSTEM$IO routine

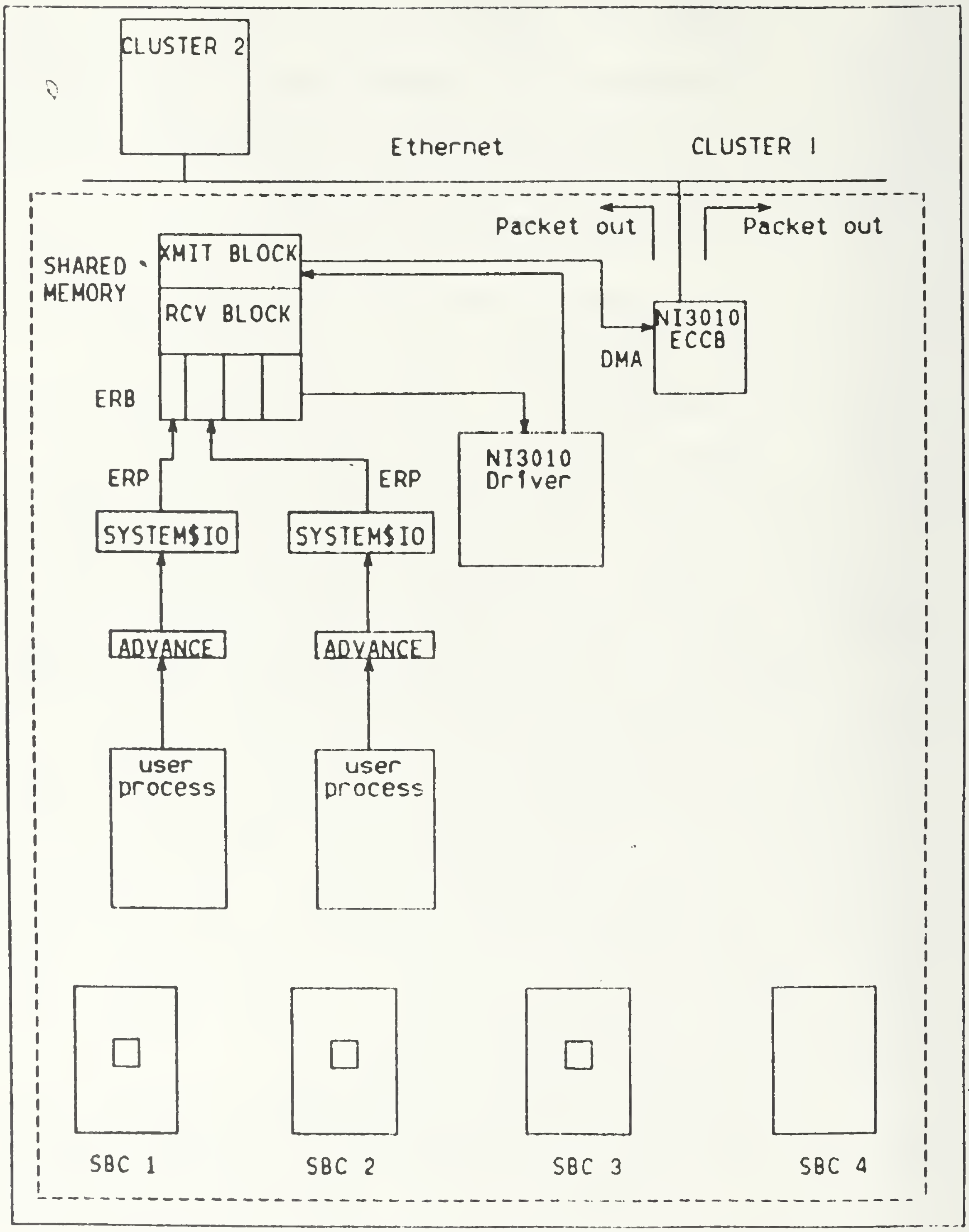


FIGURE 7 Intercluster Input/Output Processing

increments a system reserved eventcount (ERB$WRITE) to notify the NI3Ø1Ø Device Driver and Packet Processor that an ERP has been written. This "advancing" of ERB$WRITE also allows any other kernel executing the SYSTEM$IO routine to continue if it was attempting a simultaneous write to the ERB. The NI3Ø1Ø Device Driver and Packet Processor (hereafter referred to as the Driver) is a consumer of ERP's and also processes Ethernet packets received from other clusters. As a consumer of ERP's it is a system process of a cyclic nature that is scheduled in the same manner as user processes. However, this routine is dedicated to high density I/O operations, and as such is never blocked. In the highly unlikely situation where there are no ERP's to consume or packets to receive and process, the Driver idles in a 'busy wait."

Currently the only type of ERP to be processed is an 'eventcount type', whose format is shown in Figure 8. The NI3Ø1Ø Driver decodes the ERP and based on the information

| Byte 1 | Byte 2 | Byte 3 Byte 4 |
|---|---|---|
| Eventcount Type | Eventcount Name | Value |

Figure 8 Ethernet Request Packet Format

it sets up a transmit-data-block in shared memory. In fact, this block is the Ethernet packet, less the 64-bit preamble and 4-byte Frame Check Sequence (FCS). The Driver then initiates a Transmit-DMA-Done(TDD) operation to transfer the block to the transmit queue of the NI3010. The Driver follows up the TDD interrupt with a Load and Send command (29H) to the NI3010 directing it to transmit the packet over Ethernet.

Inbound packets are processed by the Driver through the Receive-Block-Available (RBA) and Receive-DMA-Done (RDD) operation sequence desribed in Chapter 3. The Driver favors outbound packets, to avoid the possibility of a bottleneck due to a "clogging up" of the ERB. When it does set up for an RBA interrupt, it will continue to the conclusion of processing the packet received. Following the DMA of the packet to the receive-data-block area in shared memory, the Driver decodes the data fields of the packet (Figure 9) and calls the appropriate MCORTEX synchronization primitives. The Driver continues to operate in this manner, determining via an eventcount value (incremented by SYSTEM$IO) whether or not an ERP exists in the ERB that needs to be processed and in the absence of one receives an inbound packet for processing.

The truly asynchronous nature of the Ethernet service should be apparent. Once SYSTEM$IO deposits an ERP, it returns immediately to the user process. The user process is not held up in its execution due to a transparent request for system input/output. The Ethernet Request Packet is the embodiment of the request, and in different forms is passed between various clusters of RTC*. It contains all the information needed to perform the operation independently of the requesting process.

## C. PACKET ROUTING ALGORITHM

Thus far, all illustrations and discussions of RTC* pertained to only two clusters, but this should not be construed as a limitation. Given that more than two clusters can exist in RTC*, some methodology must exist to route packets to as few as one and to as many as needed (up to the maximum clusters that exist).

The established convention is that no cluster will send a packet to itself. If an eventcount is advanced that requires a local update and one remote update (to one cluster) then only the local copy will be updated and only the cluster that is to receive the eventcount value will receive a packet. This clearly reduces needless packet processing at a cluster that has no interest in that eventcount, i.e., there are no producers or consumers interested in its value. Therefore an algorithm had to be developed that selectively eliminated packets from being transmitted to an inappropriate cluster.

The NI3010 has an packet addressing mode known as GROUP addressing, whereby multicast addresses can be loaded into a multicast address table onboard the NI3010. Provided this table is loaded prior to NI3010 use, any packet received that has bit 1 of the destination address field set to one (i.e., the first byte is odd) is interpreted as a multicast packet and a lookup is done in the table. If a match of the destination address is found in the table, the packet is loaded in the NI3010's receive queue. If the Driver (Figure 7) enabled an RBA interrupt, the NI3010 will issue an interrupt signifying that a packet has been received for this cluster. The Driver will then process the packet accordingly (format shown in Figure 9).

The Driver programs the NI3010 to accept GROUP addresses in its multicast table, depending on the distribution of eventcounts in RTC*. The Driver (Appendix K) has a module

DATA FIELD

```
   Byte 1    Byte 2       Byte 3  Byte 4
+------------------------------------------------+
|          |           |                         |
| Packet   |  Type     |         Value           |
|  Type    |  Name     |                         |
|  (EVC)   | (EVC ID)  |                         |
+------------------------------------------------+

        * - Packet is decoded based on
            byte 1.
```

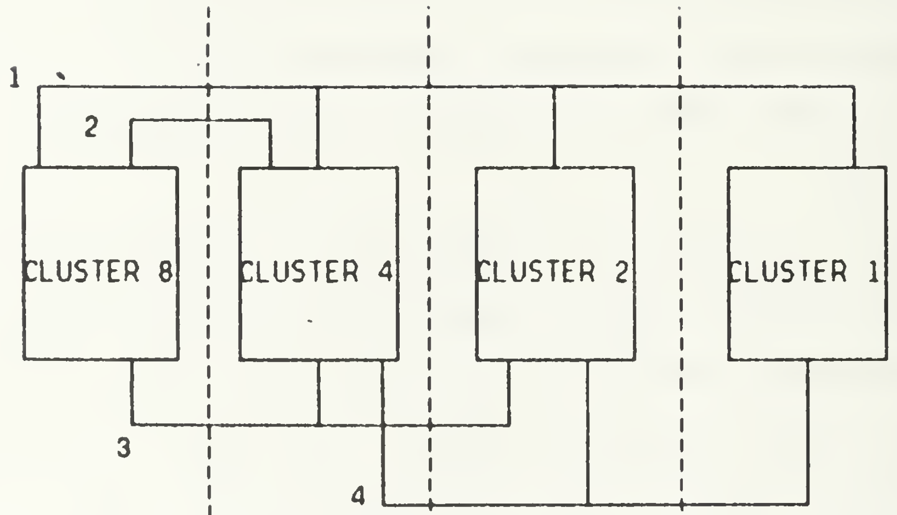
Figure 9 Eventcount Type Ethernet Packet

that reads the local cluster address and group addresses from a file called "address.dat". The local cluster address is used to set up the physical address of the NI3010 (see

[Ref. 12] for details). Any packets on Ethernet that has one of the group addresses or the physical address in the destination field is received and processed.

For packets to be transmitted over Ethernet, only the last two bytes of the destination field is programmable. This minimizes the amount of data that must be maintained and manipulated for packet addressing. The 'remote$addr' field in the EVENTCOUNT TABLE in GLOBAL memory contains the two bytes.

Figure 10 contains an example of a logical connection of clusters (they are all physically connected by Ethernet) dependent on the distributivity of the eventcounts. The lines, with numbers adjacent to them, represent a connectivity relationship of classes of data whose producers and consumers synchronize on certain eventcount values. The vertical dotted lines represent a partioning of process types and group addresses, shown below the clusters. The number in the cluster block is the physical address of each cluster. It can be seen that a producer of Type 1 data, a consumer of Type 2 data, and a consumer of Type 3 data are all present in cluster 8. A logical connectivity exists between all clusters as a result of the Type 1 data (Type 1 consumers exist at clusters 1,2, and 4). An advance by producer P1 must cause a packet to be sent to clusters 1,2, and 4.



| Process Types | Process Types | Process Types | Process Types |
|---|---|---|---|
| P1 | P2 | P3 | P4 |
| C2 | C1 | C1 | C1 |
| C3 | C3 | C4 | |
| | C4 | | |

| Group ADDR | Group ADDR | Group ADDR | Group ADDR |
|---|---|---|---|
| 0008 | 0004 | 0002 | 0001 |
| 000C | 0007 | 0007 | 0007 |
| | 000C | 0006 | |
| | 0006 | | |

P1 - Producer of Type 1 Data

C1 - Consumer of Type 1 Data

Figure 10 Ethernet Packet Routing

Consider the logical connectivity of a certain eventcount to be represented by a binary one at each cluster it connects. Therefore, for the Type 1 eventcount the 4 cluster connectivity would result in 1111 base-2 or 0F base-16. By performing an exclusive-or operation on 0FH with the producer's own physical address (08 base-16, in this case) a result of 0007H would be formed. Since consumers at clusters 1,2, and 4 are interested in Type 1 data, the NI3010 Driver must program 0007H into the multicast table. In reality the address <03-00-00-00-00-07> (6 bytes in length, first byte being odd) would appear as an entry in the table.

Continuing with this example, consider the Type 4 connectivity. The binary connectivity is 0111 and by performing an exclusive-or with the value 0001H (address of cluster 1, where the producer is present) results in 0006H. The NI3010 at clusters 2 and 4 must have <03-00-00-00-00-06> in the multicast table. All other values shown in Figure 10 are derived in an analagous manner.

The "remote$adder" field of an eventcount contains the binary connectivity discussed above. The ADVANCE procedure of MCORTEX makes a test to see if the remote$addr field is equivalent to the "local$cluster$addr" (as defined by the DEFINE$CLUSTER primitive). If they are the same then SYSTEM$IO is not called and intracluster processing continues. If they are not equivalent, then an exclusive-or operation is performed on the remote$addr field (remote$addr XOR local$cluster$addr) and the resultant two byte value and appropriate eventcount information is written to an ERP. The NI3010 Driver dequeues the ERP and forms the appropriate packet format (Figure 9), initiates the DMA operation to the NI3010, and issues the Load and Send operation.

# V. PROCESS DEVELOPMENT AND THE MCORTEX LOADER

## A. PROCESS DEVELOPMENT

### 1. PL/I-86 User Processes

Rowe [Ref. 6] is responsible for the integration of MCORTEX into the CP/M-86 environment. Although his discussion of PL/I-86 user process development is more than adequate, enough changes have been made to warrant another discussion.

Procedures written in PL/I-86 become MCORTEX processes through execution of CREATE_PROC functions. MCORTEX processes, though written, compiled, and linked as PL/I-86 procedures, are distinct processes. Each requires the state of the processor to be prepared by the MCORTEX executive prior to every entry into the process. This is accomplished transparently when making MCORTEX function calls. User-defined or built-in PL/I-86 procedures in a MCORTEX process can be accessed from within the process normally, however, a MCORTEX process must be entered through a MCORTEX function call.

KORE is the name assigned to the kernel of MCORTEX and is written in PL/M-86, and it is necessary for calls to the supervisor to meet PL/M-86 parameter passing conventions. Rowe [Ref. 6] provided mechanisms to resolve differences between simple user calls and supervisor calls.

One such mechanism is the file GATEWAY.PLI, as referred to by Rowe, and now known as the SYSDEF.PLI (for System Definitions) file. This file must be included in all programs (using the PL/I %INCLUDE directive) making calls on MCORTEX functions. The change in filename was introduced as a result of this file's multifunction role. In addition to declaring the MCORTEX functions as ENTRY values with attribute lists, the file also contains the symbolic names of eventcounts, sequencers, and pointers for shared data structures. This adds a level of security not present in previous versions of MCORTEX. The misspelling of a symbolic name will be caught by the compiler as the use of an undeclared variable.

An example of the use of pointers to access a data structure in shared memory is provided by the NI3010 Device Driver and Packet Processor. This routine performs an UNSPEC function (described in [Ref. 17 p. 72]) call to absolutely locate the Ethernet Request Block structure so that it can consume Ethernet Request Packets generated by KORE's SYSTEM$IO routine. The value appearing on the righthand side of the UNSPEC assignment statement is a symbolic name defined in the SYSDEF.PLI file. Proper static management of shared memory, with symbolic assignments, assures the integrity of user data.

Due to the limitation of pointer variables to sixteen bits in PL/I-86, some method had to be devised to allow user processes to access shared memory (outside the 64K byte range), without resorting to assembly language code to effect data moves. The ABSOLUTE feature of LINK-86 [Ref. 18] provides such an alternative. The DS register can be assigned a value (by using DATA [ABS[v]], where v represents the value) sufficiently high to allow an offset to be added to it at runtime, forming a physical address in the range 10000H - 1FFFFH (first segment shared memory). This accomplishes the desired effect. It is precisely this technique that is used in the NI3010 Driver. The Driver was linked with a value of 0800H in the LINK option file, and when added to an offset of 8000H allowed access to a based array structure called ERB (Ethernet Request Block). Note that 0800:8000 is the same as 1000:0, but the first logical address permits local data to reside in local memory and shared data in the first segment. User processes can use this same technique for interprocess communication.

MCORTEX processes that are multiplexed (multiprogrammed) on one real processor must be linked into a single CMD module. Multiprogrammed processes may share common PL/I-86 runtime routines as well as CP/M-86 utilities. However, this sharing of runtime routines and utilities presents a problem. Careful examination of the machine code of the runtime procedures and utilities revealed the fact that they are not reentrant routines. Under normal circumstances, since processes only block themselves at 'convenient' points in their execution (with the AWAIT primitive), this lack of reentrancy is not a problem. In previous versions of MCORTEX, with the preemptive interrupt 4 to signify that a process has been readied by an offboard operation, the interrupt could easily "catch" two multiplexed processes using the same non-reentrant runtime routine or utility. The change in scheduling philosophy, as discussed in the SOFTWARE SERVICES section of Chapter 3, reduces this "window of vulnerability." If a process is scheduled, via a PREEMPT operation (which still uses interrupt 4), behind a process that was blocked and using the same runtime routine or utility, the originally scheduled process's execution state could be catastrophically altered. This type of situation can be avoided through a careful distribution of user processes. That is, don't allow a process that may be readied via a PREEMPT operation to be multiplexed with a process that might possibly use the same utilities or PL/I runtime routines. If this cannot be avoided, the only remaining alternative is to write the shared code as reentrant procedures. It is anticipated that future Digital Research, Inc. language compilers and CP/M-86 operating system functions will address and resolve this lack of reentrancy. For now, it remains a problem.

MCORTEX currently expects an initialization module to be located starting at 04390H. This module is the first user process executed, and is used to create user processes only. A _system_ process written in PL/I-86 can use its intialization module to create eventcounts, sequencers, as well as creating itself. After all initializations are performed, an AWAIT('FE'B4,'0001'B4) should be executed. This puts all initialization processes on a common reserved event count thread. An ADVANCE('FE'B4) by any process will return all processors to CP/M-86 control (providing CP/M-86 is resident locally).

MCORTEX processes are written as parameterless PL/I-86 procedures. Execution of CREATE_PROC functions in the initialization module establishes a virtual processor for each process, and sets all process states to ready. The AWAIT call at the end of initializations forces a scheduling to take place. The highest priority virtual processor will be granted access to the real processor. Further scheduling is dynamically dependent on the use of MCORTEX synchronizing primitives by user processes.

Parameters required by the CREATE_PROC function include values unknown to the programmer until after all processes have been compiled and linked. This requires that dummy values be provided for the first compilation and linking. Links are performed with the MAP command option selected, since this provides information required to define user processes. A partial MAP print out for a demonstration process (full discussion in Appendix E) is shown in Table 2.

TABLE 2 - MAP FILE

```
Map for file:  C2USERS.CMD

Segments
--------

Length   Start    Stop  Align Comb  Name           Class

272D (ØØØØ:ØØØ5-2731)  BYTE  PUB  CODE            CODE
Ø5ØF (ØØØØ:Ø1ØØ-Ø6ØD)  WORD  PUB  DATA            DATA
ØØ21 (ØØØØ:Ø6ØE-Ø62E)  WORD  COM  ?CONSP          DATA
ØØ13 (ØØØØ:Ø63Ø-Ø642)  WORD  COM  ?FPBSTK         DATA
ØØ2E (ØØØØ:Ø644-Ø671)  WORD  COM  ?FPB            DATA
ØØØ2 (ØØØØ:Ø672-Ø673)  WORD  COM  ?CNCOL          DATA
ØØØ9 (ØØØØ:Ø674-Ø67C)  WORD  COM  ?FILAT          DATA
ØØØ8 (ØØØØ:Ø67E-Ø685)  WORD  COM  ?FMTS           DATA
ØØ1B (ØØØØ:Ø686-Ø6AØ)  WORD  COM  ?EBUFF          DATA
ØØØ3 (ØØØØ:Ø6A2-Ø6A4)  WORD  COM  ?ONCOD          DATA
ØØ25 (ØØØØ:Ø6A6-Ø6CA)  WORD  COM  SYSIN           DATA
ØØ28 (ØØØØ:Ø6CC-Ø6F3)  WORD  COM  SYSPRINT        DATA

Groups           Segments
------           --------

CGROUP           CODE
DGROUP           DATA     ?CONSP   ?FPBSTK    ?FPB
                 ?CNCOL   ?FILAT   ?FMTS      ?EBUFF
                 ?ONCOD   SYSIN    SYSPRINT

map for module:  C2_USERS_INIT

ØØ24  (ØØØØ:ØØØ5-ØØ28)  CODE
ØØ37  (ØØØØ:Ø1ØØ-Ø136)  DATA

map for module:  MSLORDER

ØØB5  (ØØØØ:ØØ29-ØØDD)  CODE
ØØ3B  (ØØØØ:Ø138-Ø172)  DATA

map for module:  TRKRPRT

ØØ2B  (ØØØØ:ØØDE-Ø1Ø8)  CODE
ØØ12  (ØØØØ:Ø174-Ø185)  DATA

map for module:  GATEM/T

Ø1Ø3  (ØØØØ:Ø1Ø9-Ø2ØB)  CODE
ØØØ4  (ØØØØ:Ø186-Ø189)  DATA
```

The CREATE_PROC procedure has eight actual parameters. The first two are process identification and process priority. These are BIT(8) values assigned by the software developer, with due consideration given to the module's function. Four other parameters, the CS, DS, SS, and ES register values can be determined by performing an executable load of the process CMD file under DDT86. Values displayed by DDT86 include the CS, and DS register values. As mentioned earlier, it is required that the DS, SS, and ES register values be equal for proper operation of some PL/I-86 runtime routines. Except under carefully considered circumstances, this should be the case. The remaining two parameters are pointer values obtainable from the link MAP file.

The first section of the MAP file gives a summary of all code and data segments included in the associated CMD file. Several data segments are listed in order of their occurrence in memory, from lowest offset to highest offset. The range of the last entry gives the last address offset occupied by any data segment. Higher address offsets still within the memory space of this CMD file are assigned to stack and free space structures by PL/I-86, with the system stack preceding free space. The SP value required by the CREATE_PROC function can be obtained by adding the size of the stack required to the last offset occupied by data. If another MCORTEX process stack is required, its SP can be obtained by adding its size to the SP of the previous process. The system stack can be divided as necessary by continuing in this manner. The total number of bytes occupied by MCORTEX process stacks should not exceed the number of bytes provided by PL/I-86 for the system stack.

The MAP file also contains maps of the individual modules linked into the CMD file. These maps provide data about locations of code and data segments within the larger code and data segments summarized in the segments section. The beginning address of each module is given. This offset represents the IP value for that particular module.

With all parameter values determined, the initialization process must be recompiled, and all processes relinked. The resulting CMD file can be executed in the MCORTEX environment.

### 2. Gate Module

GATEMOD.OBJ (or GATETRC.OBJ) must be linked with all user processes. It provides the object code necessary to convert user calls to the format expected by the supervisor, including addition of function codes, and padding of calls with extraneous parameters. GATEMOD uses no variable data segment of its own, and simply makes moves from user data areas to the user stack. This ensures that, so long as the user data areas involved are unshared, GATEMOD is reentrant.

GATEMOD and GATETRC both act as translators of user calls into formats required by the MCORTEX and MXTRACE supervisors respectively. The only difference in the two gate modules is the address of GATE$KEEPER in their associated KOREs. As assembly language routines called by PL/I-86 MCORTEX processes, GATEMOD or GATETRC use the established parameter passing conventions (PL/I-86 to ASM86) to build the stack structure expected by the supervisor module (PL/M-86 format), supplying function codes and padding when required. A call is then made to GATE$KEEPER. If the call is to READ or TICKET, space is reserved on the stack for the returned value. This value is popped into the BX register (PL/I-86 convention) before exiting to the calling process.

KORE functions do not guarantee the integrity of the ES register. PL/I-86 in OPTIONS (MAIN) initializations, however, establishes the ES, SS, and DS registers to be of equal value, and some runtime routines expect this relationship to be maintained. The gate modules push the ES register onto the stack on entry, and pop it before return to the calling routine, thus preserving its precall value. Entirely transparent to user processes, the ES register value is preserved throughout MCORTEX calls.

## B. MCORTEX LOADER

### 1. The Loader

Prior to Rowe's [Ref. 6] work the MCORTEX executive was assigned to the file KORE and was accessible only through utilities in the INTELLEC MDS system. This file contained all the multiprocessor operating system functions, the initial GLOBAL memory, the supervisor, the interrupt vector, and various low level functions not accessible to the user. To execute MCORTEX it was necessary to download KORE and user processes to the target system, disconnect the transfer cable, connect the target system terminals, and pass control to KORE on each processor. See [Ref. 5: Appendix A, B] for a complete description of the process.

The KORE.OPS and KORE.TRC files, now loadable under CP/M-86 through the MCORTEX and MXTRACE loaders, are derived from KORE. KORE.OPS provides no system diagnostics, whereas KORE.TRC provides CRT output to indicate the entry into MCORTEX primitives. It is expected that during the software development phases, KORE.TRC will be used to facilitate debugging. In some circumstances this may not be feasible due to the reduced speed of execution as a result of the I/O overhead.

Appendix A details the procedure used to produce KORE.OPS and KORE.TRC from KORE. Further discussion will use the terms KORE and MCORTEX to mean either KORE.OPS or

KORE.TRC and MCORTEX or MXTRACE respectively. When this generalization does not hold, the differences will be noted.

### 2. Operation of the MCORTEX Loader

MCORTEX.CMD is an executable file under the CP/M-86 operating system. Invoking MCORTEX without KORE.OPS on the default drive results in an error message and an abrupt return to CP/M-86. MXTRACE requires KORE.TRC. The loader announces that it is on line, and provides a prompt to query the interactive user whether or not GLOBAL memory should be loaded. Only the first processor activated should load GLOBAL memory. Subsequent loads of GLOBAL memory will destroy data needed by executing processors. If no initial load of GLOBAL memory is made the results are undefined.

KORE is immediately loaded with or without GLOBAL memory as directed. The load is accomplished using CP/M-86 functions, but does not use the CMD load utility. Instead, KORE is read in and positioned block at a time as required. KORE load is followed by a request for a process file name. The loader expects one file name to be entered, and results are unpredictable if a "filename.filetype" does not precede a keyboard <RETURN>. User processes are loaded using the CP/M-86 CMD load utility, and user processes must be CMD files. The entire file name must be entered including the three letter extension or filetype (.CMD). After loading the user file, the loader passes control to MCORTEX. MCORTEX initializations are performed within KORE, including creation of the IDLE and INIT processes (also MONITOR with MXTRACE), and the user initialization process is then entered. Operation after this point is determined by the user processes.

# VI. CONCLUSIONS

The principal goals of this thesis were achieved. The modifications to the previous version of MCORTEX, to allow the distribution of processes over a high speed intercluster bus, were developed and appropriately tested. Eventcount values are currently the only entities that are transferred in packet form over Ethernet. However, the framework to easily extend the distributivity of other entities is established.

From the viewpoint of user processes, access to Ethernet is gained in an entirely transparent manner. This access is truly asynchronous in the sense that a return to the requesting process occurs when an Ethernet Request Packet is written to shared memory, not when actual output of the information occurs. Provided the NI3010 Driver and Packet Processor keeps up with the I/O rate, a bottleneck will not result. The dedication of the Driver to its own real processor assures this.

The Driver software needed to distribute MCORTEX over Ethernet is device-dependent, however MCORTEX only interfaces with this routine through the convenient abstraction of an Ethernet Request Packet. Any changes in the Driver will _not_ cause an undesirable ripple effect of changes in the operating system code. This integration of harware and software is easily modified and extensible.

The creation of eventcounts and sequencers in the initialization module of a carefully tested system process provide a level of security not present before. This security is further enhanced by expanding the role of the SYSDEF.PLI file that is included in each MCORTEX process. By convention the user processes cannot alter the constant definitions present in SYSDEF. The user processes are not hostile anyway, but it will clearly not be to their advantage to alter this file. The assigning of pointers for shared structures further elevates the level of security.

The NI3010 Device Driver and Packet Processor is a basic MCORTEX system process that is highly modular, virtually self-documenting, and extensible in nature. By modifying this code and the supporting code in MCORTEX, the distribution of other entities can be achieved. The distribution of sequencers is a nontrivial matter and careful consideration must be given to the speed at which a ticket value is returned to the requesting cluster. Ethernet packets will unavoidably be queued up in NI3010 input buffers, and the speed in which they would be processed by the current Driver is fixed. A sequencer-type packet (not recognized by the current driver) would be processed immediately by the Driver, i.e., a value would be returned from the GLOBAL data of the cluster responsible for the shared resource, and an Ethernet packet would be sent out immediately.

The distribution of user shared data could similarly be achieved, with the buffering of data in the shared memory of each cluster. The synchronization on successive interations of data would be realized in the same manner as previously discussed.

The issue of packet security is a crucial one. The inherently reliable Ethernet is adequate in most instances, but a one bit error in (10 ** 8) to (10 ** 11) bits could be catastrophic enough when it occurs, so that an "acknowledging Ethernet" may have to be developed. Enough adequate testing has not been conducted in the AEGIS Simulation Laboratory to draw any conclusions in this area.

The lack of reentrancy in runtime code and CP/M-86 utilities is an issue that needs to be more actively addressed. A "LARGE" PL/I-86 compiler is under development by Digital Research, Inc. that should resolve the reentrancy problem and the limited range (64K bytes) of pointer variables. This product should be available in January 1985. In addition to solving the aforementioned problems, the "LARGE" compiler will also sever the umbilical cord between the ISIS-II and CP/M-86 operating systems. MCORTEX development can then continue in PL/I-86 instead of PL/M-86. MCORTEX will then evolve rapidly and consistently with increasingly more complex user processes.

# APPENDIX A

## ISIS-II TO CP/M-86 TRANSFER

I. PRE-POWER-ON CHECKS

A. SBC configured for CP/M-86 cold boot is in MULTIBUS odd slot and no other clock master SBC is installed.

B. REMEX controller is in MULTIBUS, and properly connected to REMEX drive.

C. If MICROPOLIS hard disk is to be used, ensure that it is connected to clock master SBC.

D. Ensure 32K shared memory module is installed.

E. Connect RS232 transfer cable between J2 on SBC, and 2400 baud CRT port of the MDS system. If this cable has a 'null modem' switch on it, set it to "null modem". This transposes wires 2 and 3. The switch may be marked "computer to computer" and "computer to terminal". Set to "computer to computer".

F. Connect any CRT to the 9600 baud TTY port of the MDS system. Ensure CRT is set to 9600 baud.

G. A CRT will be connected to the SBC after the loading is completed, and should have an RS232 cable hooked to the serial port. The CRT connection should lead to a flat 25 wire ribbon and J2 connector so it can eventually be hooked to the SBC's serial port.

## II. POWER ON PROCEDURES

A. Turn the power-on key to ON position at MULTIBUS frame.

B. Press RESET near power-on key.

C. If needed apply power to MICROPOLIS hard disk.

D. Apply power to REMEX disk system. After system settles, put START/STOP switch in START position. Following a lengthy time-out period, the READY light on the front of the REMEX disk system will illuminate, and the system is ready.

E. Insert the boot disk into drive B.

F. Apply power to the CRT.

G. Power up the MDS disk drive.

H. Power up the MDS terminal.

I. Turn power-on key to ON at MDS CPU.

## III. BOOT UP MDS

A. Place diskette with executable modules and SPC861 in drive Ø.

B. Push upper part of boot switch in (It will remain in that position).

C. Press reset switch and then release it.

D. When the interrupt light #2 lights on the front panel, press space bar on the console device.

F. Reset the boot switch by pushing the lower part of the switch.

F. ISIS-II will announce itself and give the '-' prompt.

IV. LOAD KORE

A. At MDS console, type 'SBC861<CR>'.

B. IF "*CONTROL*" appears, SBC was not able to set its baud rate. Press RESET on MULTIBUS frame and try again.

C. If 'Bad EMDS connection' appears, you will not be able to continue. Check connectons. Make sure diskette is not write protected. Push RESET at frame. Try again.

D. SBC861 will announce itself and prompt with ".".

E. Type 'I KORE<cr>'. Wait for ".". At this point the KORE module has been loaded into the SBC memory, and into the common memory board.

V. SAVING KORE TO CP/M-86 FILE

A. Leaving the SBC861 process active on the MDS system, disconnect the RS232 J2 connector at the SBC, and connect the terminal prepared earlier.

B. At the newly connected terminal type "GFFD4:4<cr>". The CRT will not echo this entry. Respond to the cues that follow as required until CP/M-86 is up.

C. Now enter DDT86. At this point KORE, CP/M-86, and DDT86 all are resident in the SBC memory and in the 32K shared memory board.

D. Using DDT86 commands, reposition the parts of KORE required so that the code can be saved into one file. Data necessary to determine the initial locations of the code is found in KORE.MP2. The DDT86 instructions used for the current KORE.OPS and KORE.TRC files follows:

*** KORE.OPS ***

MB70:0,1000.480:0 *** Move, starting at address B70:0, 1000 bytes of code (main part of KORE) to new start address 480:0.

M439:0,80,580:0 *** Move, starting at address 439:0, 80 bytes of code (initialization module) to new start address 580:0 (following main part as moved above).

ME530:0,800,588:0 *** Move, starting at address E530:0, 800 bytes of code (GLOBAL memory) to new start address 588:0 (following initialization module).

WKORE.OPS,480:0,1880 *** Write to the default disk a file called KORE.OPS starting at address 480:0 and containing 1880 bytes.

*** KORE.TRC ***

MAC0:0,1000,480:0 *** Move, starting at address AC0:0, 1000 bytes of code (main part of KORE) to new starting address 480:0.

M439:0,80,640:0 *** Move, starting at address 439:0, 80 bytes of code (initialization module) to new starting address 640:0 (following main part of KORE) .

ME530:0,800.648:0 *** Move, starting at address E530:0, 800 bytes of code (GLOBAL memory) to new starting address 648:0 (following main KORE & initialiaztion module).

WKORE.TRC,480:0,2480   *** Write to the default disk a file called KORE.TRC starting at address 480:0 and containing 2480 bytes.

NOTE: The main KORE module, the initialization module, and GLOBAL memory are located to separate parts of the SBC by the MCORTEX loader. The system used requires that these modules be saved into the file in 128 byte blocks. Further, any change in the number of 128 byte blocks occupied by each must be reflected in the MCORTEX loader code.

# APPENDIX B

## DEBUGGING TECHNIQUES

DDT86 [Ref. 13] is the primary debugging tool used in software product development in the AEGIS Simulation Laboratory. This debugger allows the user to test and debug programs interactively in a CP/M-86 environment. Far from being a high level debugging tool, DDT86 nevertheless provides the user with the ability to interactively enter assembly language statements, display the contents of memory, trace program execution, and utilize other commands to provide software development assistance.

The use of DDT86 in the development of the NI3010 Device Driver and Packet Processor was invaluable. Ethernet Request Packets could be interactively written to shared memory and the response of the Driver was easily monitored from the same terminal. Breakpoints can be set in processes and the execution of a single board computer will continue until the breakpoint is reached. A process can block and when scheduled next, by a kernel of MCORTEX, the CPU will break at the setpoint.

A particularly valuable feature, that unfortunately is unavailable in DDT86, is that of a **watchpoint**. A **watchpoint** is defined here as a location that a debugger would monitor and inform the user when an executing program has made an attempt to execute an instruction at that location. This feature can be emulated under DDT86 by using the "A" command (enter assembly language statements) to enter an INT 3 (interrupt 3) command. What the user does not get, however, is a history of the instructions that got the CPU to this execution point. In a single step trace this is not a problem, but execution at near real-time is. In highly modular software, such as MCORTEX, the single step trace through levels of procedure calls can be an extremely laborious task.

In situations where the state of the CPU does not appear consistent with the executing software, and the reliability of the hardware is questionable, there are few acceptable alternatives to using a digital logic analyzer. The Paratronics 532 is the logic analyzer used extensively in the AEGIS Simulation Laboratory.

# APPENDIX C

## MCORTEX LOADER

This file is assembled using the RASM86 assembler [Ref. 18]. After linking, when invoked as a transient command from the CCP level of CP/M-86, this file will interactively allow the loading of a CMD file containing a MCORTEX process or multiplexed MCORTEX processes. Only the first real processor entering the MCORTEX environment is to specify that GLOBAL data is to be loaded. Conditional assembly features pervade this code to allow either MCORTEX or MXTRACE (the diagnostic version) to be loaded. The conditional switch is called "MCORTEX", which is set equal to one (or TRUE) when the MCORTEX version of the loader is to be assembled. The use of the MCORTEX or MXTRACE LINK86 input option files (APPENDIX F) determine which transient command is generated.

```
;*************************************************************/
;* MCORTEX / MXTRACE File TEX/TRC.A86  Brewer 24 AUG 84    */
;*------------ ---------- ------------------ ----------------*/
;* This program loads the MCORTEX operating system from    */
;* disk into the current CP/M environment.  The system     */
;* memory space is reserved using CP/M memory management   */
;* functions.  Since INITIALPROC must be overwritten by    */
;* the user INITIALPROC, the memory it occupies is not     */
;* reserved.  The portions loaded into the interrupt       */
;* area and into shared memory (ie. GLOBALMODULE) are in   */
;* areas not managed by CP/M and are thus protected from   */
;* user overwrite when using PLI CMD files.  Conditional   */
;* assemblies allow assembly of either MCORTEX or MXTRACE*/
;* depending on the value assigned to MCORTEX at the       */
;* beginning of the code.  Nine such conditional           */
;* assembly statements are included.                       */
;*************************************************************/

            DSEG
            ORG  0000H
;*** MCORTEX / MXTRACE SELECTION ****************************/

MCORTEX                     EQU 0 ;*** SET TO ZERO FOR
                                  ;*** MXTRACE

;*** ADDRESS CONSTANTS **************************************/

FCB                         EQU 005CH       ;*** FILE CONTROL
FCB_NAME                    EQU 005DH       ;*** BLOCK
FCB_EXTENT                  EQU 0068H
FCB_CR                      EQU 007CH

INT_ADD_CS                  EQU 0011H     ;*** INTERRUPT CODE
INTRPT_OFFSET               EQU 0033H     ;*** SEGMENT AND
IF MCORTEX
INTRPT_CS                   EQU 0C4BH     ;*** VECTOR
ELSE
INTRPT_CS                   EQU 0C4FH     ;#### 1 #### <----
ENDIF

;*** PURE NUMBER CONSTANTS **********************************/

EIGHTH_K                    EQU 0080H
IF MCORTEX
NUM_KORE_BLOCKS             EQU 0020H
ELSE
NUM_KORE_BLOCKS             EQU 0038H     ;#### 2 #### <----
ENDIF
NUM_GLOBAL_BLOCKS           EQU 0010H

ASCII_0                     EQU '0'
```

```
ASCII_9                     EQU '9'
ASCII_A                     EQU 'A'
ASCII_Z                     EQU 'Z'
SLASH                       EQU '/'
COLON                       EQU ':'
SPACE                       EQU ' '
PERIOD                      EQU '.'
CR                          EQU 000DH
LF                          EQU 000AH


;*** CONTROL TRANSFER CONSTANTS ***************************/

IF MCORTEX
KORE_SP                     EQU 0075H
KORE_SS_VAL                 EQU 0C55H
KORE_DS_VAL                 EQU 0C49H
ELSE
KORE_SP                     EQU 0075H   ;#### 3 #### <-----
KORE_SS_VAL                 EQU 0C5BH   ;#### 4 #### <-----
KORE_DS_VAL                 EQU 0C2CH   ;#### 5 #### <-----
ENDIF

;*** CP/M FUNCTION CONSTANTS ******************************/

CPM_BDOS_CALL               EQU 224
SYSTEM_RESET                EQU 0000H
CONSOLE_OUTPUT              EQU 0002H
READ                        EQU 000AH
PRINT_STRING                EQU 0009H
OPEN_FILE                   EQU 000FH
READ_SEQUENTIAL             EQU 0014H
SET_DMA_OFFSET              EQU 001AH
SET_DMA_BASE                EQU 0033H
ALLOC_MEM_ABS               EQU 0038H
FREE_ALL_MEM                EQU 003AH
PROGRAM_LOAD                EQU 003BH
NOT_FOUND                   EQU 00FFH

;*** MESSAGES *********************************************/

IN_STRING                   DB 15
                            RB 16

NO_FILE_MSG DB 'KORE NOT ON DEFAULT DRIVE$'
NO_IN_FILE_MSG DB 'INPUT FILE NOT ON DESIGNATED DRIVE$'
NO_MEMORY_MSG DB 'UNABLE TO ALLOCATE MEMORY SPACE FOR'
              DB ' MCORTEX$'
FILE_FORM_ERR_MSG DB 'INCORRECT FILE FORMAT - TRY AGAIN$'

START_MSG DB 'MCORTEX SYSTEM LOADER *** ON LINE$'
```

```
P_NAME_MSG DB CR,LF,LF,'ENTER PROCESSOR FILE NAME:',CR,LF
           DB '$'
GLOBAL_Q_MSG DB CR,LF,LF,'LOAD GLOBAL MEMORY?',CR,LF
GM2_MSG DB ' 'Y' TO LOAD, <RETURN> IF NOT',CR,LF,'$'

;*** MCORTEX RELOCATION VARIABLES ***************************/

;*** CAUTION *** CAUTION *** CAUTION *** CAUTION **********/
;*** The following five lines of code should not be    ***/
;*** separated as this program assumes they will be    ***/
;*** found in the order shown.  The code is used for    ***/
;*** memory allocation and as a pointer to KORE.        ***/
;*** CAUTION *** CAUTION *** CAUTION *** CAUTION **********/

KORE_START                 DW  0030H              ;*** CAUTION
IF MCORTEX
KORE1_BASE                 DW 0B70H               ;*** CAUTION
ELSE
KORE1_BASE                 DW 0AC0H     ;#### 6 #### <-----
ENDIF
KORE                   EQU DWORD PTR KORE_START ;*** CAUTION
IF MCORTEX
KORE1_LENGTH               DW 0100H               ;*** CAUTION
ELSE
KORE1_LENGTH               DW 01C0H     ;#### 7 #### <-----
ENDIF
KORE1_M_EXT                DB 0                   ;*** CAUTION

IF MCORTEX
KORE_NAME                  DB 'KORE    OPS'
ELSE
KORE_NAME                  DB 'KORE    TRC' ;### 8 ### <--
ENDIF

KORE2_BASE                 DW 0E530H ;*** GLOBAL MEMORY

INTERRUPT_VECTOR           DW INTRPT_OFFSET,INTRPT_CS
INT_VECTOR_ADD             DW INT_ADD_CS

INIT_OFFSET         DW 0000H  ;*** INITIALIZATION
INIT_BASE           DW 0439H  ;*** ROUTINE PARAMETERS
IF MCORTEX
INIT_DS_SEG         DW 0C65H  ;*** FOR DYNAMIC ASSIGNMENT
ELSE
INIT_DS_SEG         DW 0C6BH        ;#### 9 #### <--------
ENDIF
INIT_DS_OFFSET      DW 0068H  ;*** WHEN USER INITIALIZATION
INIT_IP_OFFSET      DW 0074H  ;*** IS INDICATED

;*** CONTROL TRANSFER VARIABLES ****************************/

KORE_SS                    DW KORE_SS_VAL
```

```
KORE_DS                          DW KORE_DS_VAL

;*** START CODE SEGMENT ***********************************/

MCORTEX_LOADER CSEG

CALL CLR_SCREEN            ;*** SCREEN CONTROL & LOG ON
CALL MCORTEX_LOAD          ;*** MESSAGES
CALL CLR_SCREEN            ;***

CLD                        ;*** INITIALIZATION
PUSH AX                    ;***

;*** GET LOAD GLOBAL INDICATOR *****************************/

CALL IN_GLOBAL               ;*** ASK IF GLOBAL TO BE LOADED
MOV DX,OFFSET IN_STRING ;*** GET BUFFER LOCATION
MOV CL,READ                  ;*** CP/M PARAMETER
INT CPM_BDOS_CALL            ;*** GET INDICATER

;*** GENERATE KORE FILE CONTROL BLOCK *********************/

GEN_KORE_FCB:
MOV BX,1Ø                    ;*** MOVE 11 CHARACTERS
MOV SI,OFFSET KORE_NAME ;*** POINT TO KORE NAME
MOV DI,FCB_NAME              ;*** POINT TO FCB NAME
MOV_KORE:
MOV AL,[SI+BX]               ;*** GET CHARACTER
MOV [DI+BX],AL               ;*** STORE CHARACTER
DEC BX
JGE MOV_KORE

;*** OPEN KORE.OPS FILE ON DEFAULT DISK *******************/

OPEN_KORE:
MOV CL, OPEN_FILE                     ;*** CP/M PARAMETER
MOV DX,FCB                            ;*** CP/M PARAMETER
INT CPM_BDOS_CALL                     ;*** OPEN FILE
CMP AL,NOT_FOUND                      ;*** FILE FOUND?
JNE PROCESS_KORE                      ;*** FILE FOUND! CONTINUE
JMP NO_FILE                           ;*** GO INDICATE ERROR
PROCESS_KORE:
MOV DI,Ø
MOV FCB_CR[DI],DI                     ;*** START WITH REC ZERO

;*** RESERVE MEMORY ***************************************/

MOV CL,FREE_ALL_MEM           ;*** CP/M PARAMETER
INT CPM_BDOS_CALL             ;*** FREE ALL MEMORY
MOV CL,ALLOC_MEM_ABS          ;*** CP/M PARAMETER
MOV DX,OFFSET KORE1_BASE      ;*** CP/M PARAMETER
INT CPM_BDOS_CALL             ;*** ALLOCATE MEMORY
```

```
CMP AL, NOT_FOUND               ;*** MEMORY AVAILABLE?
JNE LOAD_MCORTEX                ;*** MEMORY AVAILABLE! CONTINUE
JMP NO_MEMORY_ALLOC             ;*** GO INDICATE ERROR

;*** LOAD MCORTEX CODE **************************************/

LOAD_MCORTEX:
MOV DI,0                          ;*** SET DEST. OFFSET
MOV BP,NUM_KORE_BLOCKS            ;*** SET BLOCK COUNTER
MOVE_KORE_LOOP:
MOV DX,FCB                        ;*** CP/M PARAMETER
MOV CL,READ_SEQUENTIAL            ;*** CP/M PARAMETER
INT CPM_BDOS_CALL                 ;*** READ IN 128 BYTES
MOV ES,KORE1_BASE                 ;*** SET DESTINATION SEGMENT
MOV CX,EIGHTH_K                   ;*** SET BYTE COUNT
MOV SI,CX                         ;*** SET SOURCE OFFSET
REP  MOVSB                        ;*** MOVE 128 BYTES
DEC BP                            ;*** DEC BLOCKS TO MOVE
JNZ MOVE_KORE_LOOP                ;*** IF NOT DONE, DO AGAIN

;*** LOAD INITIALIZATION MODULE ***************************/

MOV DI,INIT_OFFSET                ;*** SET DEST. OFFSET
MOV DX,FCB                        ;*** CP/M PARAMETER
MOV CL,READ_SEQUENTIAL            ;*** CP/M PARAMETER
INT CPM_BDOS_CALL                 ;*** READ IN 128 BYTES
MOV ES,INIT_BASE                  ;*** SET DESTINATION SEGMENT
MOV CX,EIGHTH_K                   ;*** SET BYTE COUNT
MOV SI,CX                         ;*** SET SOURCE OFFSET
REP MOVSB                         ;*** MOVE 128 BYTES

;*** LOAD GLOBAL MEMORY *************************************/

CMP IN_STRING+1,0H                ;*** SHOULD GLOBAL BE LOADED?
JZ INSTALL_INTERRUPT              ;*** IF NOT, SKIP LOAD
MOV DI,0                          ;*** SET DEST. OFFSET
MOVE_GLOBAL_LOOP:
MOV DX,FCB                        ;*** CP/M PARAMETER
MOV CL,READ_SEQUENTIAL            ;*** CP/M PARAMETER
INT CPM_BDOS_CALL                 ;*** READ 128 BYTES
TEST AL,AL                        ;*** NO MORE DATA?
JNZ INSTALL_INTERRUPT             ;*** NO, SO GO ON
MOV ES,KORE2_BASE                 ;*** SET DEST. SEGMENT
MOV CX,EIGHTH_K                   ;*** SET BYTE COUNT
MOV SI,CX                         ;*** SET SRC. OFFSET
REP  MOVSB                        ;*** MOVE 128 BYTES
JMP MOVE_GLOBAL_LOOP              ;*** IF NOT DONE, DO AGAIN

;*** INITIALIZE INTERRUPT VECTOR ***************************/

INSTALL_INTERRUPT:
MOV ES,INT_VECTOR_ADD               ;*** SET DESTINATION SEGMENT
MOV DI,0                            ;*** SET DEST. OFFSET
MOV SI,OFFSET INTERRUPT_VECTOR ;*** SRC. OFFSET
MOV CX,2                            ;*** 2 WORDS TO MOVE
REP MOVS AX,AX                      ;*** MOV TWO WORDS

;*** READ IN A FILE NAME ***********************************/

READ_A_NAME:
CALL PROCESSOR_NAME               ;*** MSG TO INPUT A FILE NAME
MOV DX,OFFSET IN_STRING           ;*** DX <-- BUFFER LOCATION
MOV CL,READ                       ;*** CPM PARAMETER
INT CPM_BDOS_CALL                 ;*** GET A FILE NAME

;*** SET FCB DRIVE DESIGNATION ******************************/

MOV DI,0               ;*** SET DESTINATION INDEX TO ZERO

CMP IN_STRING+3,COLON ;*** IS DRIVE DESIGNATED?
JE SET_DRIVE           ;*** IF YES, PUT DRIVE IN FCB
MOV FCB[DI],DI         ;*** SET DEFAULT DRIVE
MOV SI,2               ;*** 3RD POSIT IN_STRING, IS 1ST LETTER
JMP FORM_FCB

SET_DRIVE:
MOV AL,IN_STRING+2 ;*** GET DRIVE LETTER
AND AL,5FH             ;*** CONVERT TO UPPER CASE
SUB AL,40H             ;*** CONVERT TO A BINARY NUMBER
MOV FCB[DI].AL         ;*** SET DRIVE
AND AL,0F0H            ;*** LIMIT LINE DRIVE TO A THROUGH O
TEST AL,AL
JNZ INPUT_ERROR_B
MOV SI,4               ;*** 5TH POSIT IN_STRING IS 1ST LETTER

;*** INITIALIZE FILE CONTROL BLOCK *************************/

FORM_FCB:
MOV BX,0AH                 ;*** FILL FCB NAME WITH SPACES
MOV AL,SPACE               ;***
FILL_SPACES:
MOV FCB_NAME[BX],AL        ;***
DEC BX                     ;***
JGE FILL_SPACES            ;***

MOV FCB_CR[DI],DI          ;*** NEW FILE CURRENT RECORD IS ZERO
MOV FCB_EXTENT[DI],DI      ;*** NEW FILE CURRENT EXTENT IS ZERO

;*** INSTALL FILE CONTROL BLOCK NAME **********************/

NAME_LOOP:
MOV AL,IN_STRING[SI]     ;*** GET A CHARACTER
CMP AL,PERIOD            ;*** START TYPE ?
JNE FCB_CONT_1           ;*** IF NO, CONTINUE
```

```
MOV DI,8                ;*** IF YES, DJUST DESTINATION
JMP FCB_CONT_2          ;*** AND CONTINUE
FCB_CONT_1:
CALL VALID_INPUT        ;*** CHECK FOR LETTER OR NUMBER
TEST AX,AX              ;***
JE INPUT_ERROR_B        ;***
MOV FCB_NAME[DI],AL     ;*** MOVE CHARACTER INTO FCB
MOV AX,SI               ;*** IS THIS LAST CHARACTER?
CMP IN_STRING+1,AL      ;***
JB OPEN_PROCESSOR       ;*** IF YES, LOAD THE FILE
INC DI                  ;*** IF NO, ADJUST FOR NEXT LETTER
FCB_CONT_2:
INC SI                  ;*** AND GO AGAIN
JMP NAME_LOOP           ;***

EXIT_ROUTINE_B:
JMP EXIT_ROUTINE        ;*** BRIDGE TO EXIT ROUTINE
INPUT_ERROR_B:
JMP INPUT_ERROR        ;*** BRIDGE TO INPUT_ERROR




;*** OPEN THE PROCESSOR FILE *******************************/

OPEN_PROCESSOR:
MOV DX,FCB              ;*** CP/M PARAMETER
MOV CL,OPEN_FILE        ;*** CP/M PARAMETER
INT CPM_BDOS_CALL       ;*** OPEN THE FILE
CMP AL, NOT_FOUND       ;*** WAS FILE ON DISK
JNE LOAD_PROCESSOR      ;*** IF YES, GO LOAD THE FILE
JMP NO_INPUT_FILE       ;*** IF NO, SIGNAL ERROR
LOAD_PROCESSOR:
MOV DX,FCB              ;*** CP/M PARAMETER
MOV CL,PROGRAM_LOAD     ;*** CP/M PARAMETER
INT CPM_BDOS_CALL       ;*** LOAD THE FILE
                        ;*** DATA SEGMENT IN AX

;*** SET UP THE INITIALIZATION STACK ***********************/

;*** CAUTION *** CAUTION *** CAUTION *** CAUTION *********/
;*** This code is highly dependent upon Input of PL/I  ***/
;*** CMD file with CS header first and data header     ***/
;*** second.  This is the normal situation and should  ***/
;*** cause no difficulty.  Also this code is highly    ***/
;*** dependent upon the location of the initialization ***/
;*** module stack and the location of the DS and IP    ***/
;*** values within that stack.  Changes in stack       ***/
;*** location or organization should be reflected here.***/
;*** CAUTION *** CAUTION *** CAUTION *** CAUTION *********/

EXIT_ROUTINE:
```

```
MOV ES,INIT_DS_SEG            ;*** POINT TO INIT STACK
MOV BX,INIT_DS_OFFSET         ;*** POINT TO DS ON STACK
MOV ES:[BX],AX                ;*** INSTALL NEW INIT DS
MOV DX,Ø                      ;*** SET NEW IP VALUE
MOV BX,INIT_IP_OFFSET         ;*** POINT TO IP ON STACK
MOV ES:[BX],DX                ;*** INSTALL NEW INIT IP
MOV CL,SET_DMA_BASE           ;*** CP/M PARAMETER
MOV DX,AX                     ;*** SET BASE PAGE
INT CPM_BDOS_CALL             ;*** SET DMA BASE
MOV CL,SET_DMA_OFFSET         ;*** CP/M PARAMETER
MOV DX,EIGHTH_K               ;*** GET OFFSET
INT CPM_BDOS_CALL             ;*** SET DMA OFFSET

;*** TRANSFER CONTROL TO MCORTEX **************************/

MOV SP,KORE_SP                         ;*** KORE STACK POINTER
MOV BP,SP                              ;*** KORE STACK BASE
MOV SS,KORE_SS                         ;*** KORE STACK SEGMENT
MOV AX,DS                              ;*** GET DATA SEGMENT
MOV ES,AX                              ;*** POINT ES TO DS
MOV DS,KORE_DS                         ;*** KORE DATA SEGMENT
JMPF ES:KORE                           ;*** JUMP TO MCORTEX

;*** VALID CHARACTER FOR FILE NAME CHECK *****************/

VALID_INPUT:
CMP AL,SLASH
JE  IS_VALID
CMP AL,ASCII_Ø          ;*** IS THE CHARACTER A NUMBER
JB NOT_VALID            ;***
CMP AL,ASCII_9          ;***
JBE IS_VALID            ;***
AND AL,5FH              ;*** CONVERT CHARACTER TO UPPER CASE
CMP AL,ASCII_A          ;*** IS THE CHARACTER A LETTER
JB NOT_VALID            ;***
CMP AL,ASCII_Z          ;***
JBE IS_VALID            ;***
NOT_VALID:
MOV AX,Ø                ;*** INDICATE BAD CHARACTER
IS_VALID:
RET                     ;*** CHARACTER OK

;*** ABORT MESSAGES **************************************/

NO_FILE:
CALL CLR_SCREEN
MOV DX,OFFSET NO_FILE_MSG     ;*** PTR TO MSG
JMP MSG_OUTPUT                ;*** PUT MSG

NO_MEMORY_ALLOC:
CALL CLR_SCREEN
MOV DX,OFFSET NO_MEMORY_MSG   ;*** PTR TO MSG
```

```
MSG_OUTPUT:
MOV CL,PRINT_STRING             ;*** CP/M PARAMETER
INT CPM_BDOS_CALL               ;*** SEND CHAR TO CONSOLE
CALL CLR_SCREEN
MOV CL,SYSTEM_RESET             ;*** CP/M PARAMETER
MOV DL,0                        ;*** RELEASE MEMORY
INT CPM_BDOS_CALL               ;*** EXIT TO CP/M

;*** SCREEN CONTROL *************************************/

CLR_SCREEN:
MOV CL,CONSOLE_OUTPUT    ;*** ISSUE CARRIAGE RETURN
MOV DL,CR                ;***
INT CPM_BDOS_CALL        ;***
MOV DI,0CH               ;*** ISSUE 12 LINE FEEDS
LINE_FEED:
MOV DL,LF                ;***
MOV CL,CONSOLE_OUTPUT    ;***
INT CPM_BDOS_CALL        ;***
DEC DI                   ;***
JNE LINE_FEED            ;***
RET

SEND_MSG:
MOV CL,PRINT_STRING      ;*** CP/M PARAMETER
INT CPM_BDOS_CALL        ;*** PRINT A STRING TO CONSOLE
RET

;*** NON ABORT MESSAGES *********************************/

MCORTEX_LOAD:
MOV DX,OFFSET START_MSG
CALL SEND_MSG
RET

PROCESSOR_NAME:
MOV DX,OFFSET P_NAME_MSG
CALL SEND_MSG
RET

IN_GLOBAL:
MOV DX,OFFSET GLOBAL_O_MSG
CALL SEND_MSG
RET

INPUT_ERROR:
CALL CLR_SCREEN
MOV DX,OFFSET FILE_FORM_ERR_MSG
JMP EXIT_ERR

NO_INPUT_FILE:
```

```
CALL CLR_SCREEN
MOV DX,OFFSET NO_IN_FILE_MSG
EXIT_ERR:
CALL SEND_MSG
CALL CLR_SCREEN
JMP READ_A_NAME

END
```

# APPENDIX D

## GATE MODULE SOURCE CODE

SYSDEF.PLI and GATEM/T.A86 files are contained in this appendix. PL/I-86 entry variables in SYSDEF.PLI provide a "gateway" to the MCORTEX (kernel) supervisor via GATEMOD or GATETRC. Also contained in SYSDEF.PLI are constant (or symbolic) definitions that are used by the demonstration processes contained in Appendix E. Note that system reserved constants, used by MCORTEX kernels and the NI3010 Driver and Packet Processor are also contained in this file.

GATEM/T.A86 is assembled, and as a relocatable object file, is linked with MCORTEX processes to set up the PL/I-86 to PL/M-86 parameter passing interface.

A conditional assembly switch "GATEMOD" allows for assembly of a GATEMOD or GATETRC version.

```
/*************************************************************/
/*************************************************************/
/** SYSDEF FILE: SYSDEF.PLI   David J. BREWER  1 SEP 84 **/
/**=======================================================**/
/** This section of code is given as a PLI file to be    **/
/** %INCLUDE'd with MCORTEX user programs.  ENTRY         **/
/** declarations are made for all available MCORTEX      **/
/** functions.                                           **/
/*************************************************************/
/*************************************************************/

    DECLARE

      advance ENTRY (BIT (8)),
        /* advance (event_count_id) */

      await ENTRY (BIT (8), BIT (16)),
        /* await (event_count_id, awaited_value) */

      create_evc ENTRY (BIT (8)),
        /* create_evc (event_count_id) */

      create_proc ENTRY (BIT (8), BIT (8),
                         BIT (16), BIT (16), BIT (16),
                         BIT (16), BIT (16), BIT (16)),
        /* create_proc (processor_id, processor_priority,*/
        /*             stack_pointer_highest, stack_seg, ip */
        /*             code_seg, data_seg, extra_seg)       */

      create_seq ENTRY (BIT (8)),
        /* create_seq (sequence_id) */

      preempt ENTRY (BIT (8)),
        /* preempt (processor_id) */

      read ENTRY (BIT (8)) RETURNS (BIT (16)),
        /* read (event_count_id) */
        /* RETURNS current_event_count */

      ticket ENTRY (BIT (8)) RETURNS (BIT (16)),
        /* ticket (sequence_id) */
        /* RETURNS unique_ticket_value */

      define_cluster ENTRY  (bit (16)),

        /* define_cluster (local_cluster_address) */

      distribution_map ENTRY (bit (8), bit (8), bit (16)),

     /* distribution_map (distribution_type, id,
                          cluster_addr)          */
    add2bit16 ENTRY(BIT(16),BIT(16)) RETURNS (BIT(16));
      /* add2bit16 ( a_16bit_#, another_16bit_#) */
      /* RETURNS a_16bit_# + another_16bit_#     */


 %replace

   /*-------------------------------------------------
              ***  EVC$ID's ***

          (1) USER                              */

      TRACK_IN                  by '01'b4,
      TRACK_OUT                 by '02'b4,
      MISSILE_ORDER_IN          by '03'b4,
      MISSILE_ORDER_OUT         by '04'b4,

        /* (2) SYSTEM                                  */

      ERB_READ                  by 'fc'b4,
      ERB_WRITE                 by 'fd'b4,

 /*----------------------------------------------------
             ***   SEQUENCER NAMES    ***

          (1) USER                                    */


        /* (2) SYSTEM  */

      ERB_WRITE_REQUEST         by 'ff'b4,

 /*----------------------------------------------------

          ***  SHARED VARIABLE POINTERS   ***

          (1) USER                                    */


      /*   (2) SYSTEM  */

      block_ptr_value           by '8000'b4,
      xmit_ptr_value            by '8078'b4,
      rcv_ptr_value             by '8666'b4,

      END_RESERVE               by 'FFFF'b4;


;***************************************************************/
;* GATEMOD / GATETRC   File GATEM/T.a86  BREWER 1 SEP  84 */
```

```
;*---------------------------------------------------------------*/
;* This module is given to the user in obj form to link        */
;* with his initial and process modules.  Any changes to        */
;* user services available from the OS must be reflected        */
;* here.  In this way the user need not be concerned with       */
;* actual GATEKEEPER services codes.  Two lines of code         */
;* are contained in conditional assembly statements and         */
;* control the output to be GATEMOD or GATETRC depending        */
;* on the value of GATEMOD at the code start.                   */
;*---------------------------------------------------------------*/
;* This module reconciles parameter passing anomalies           */
;* between MCORTEX (written in PL/M) and user programs          */
;* (written in PL/I).                                           */
;*---------------------------------------------------------------*/
;* All calls are made to the GATEKEEPER in LEVEL2 of the        */
;* OS.  The address of the GATEKEEPER must be given below.*/
;*---------------------------------------------------------------*/
;* The ADD2BIT16 function does not make calls to MCORTEX. */
;* It's purpose is to allow the addition of two unsigned        */
;* 16 bit numbers from PL/I programs.                           */
;****************************************************************/

DSEG

GATEMOD EQU 0 ;*** SET TO ZERO FOR GATETRC
              ;*** SET TO ONE FOR GATEMOD

PUBLIC ADVANCE          ;*** THESE DECLARATIONS MAKE THE
PUBLIC AWAIT            ;*** GATEKEEPER FUNCTIONS VISIBLE
PUBLIC CREATE_EVC       ;*** TO EXTERNAL PROCESSES
PUBLIC CREATE_PROC
PUBLIC CREATE_SEQ
PUBLIC PREEMPT
PUBLIC READ
PUBLIC TICKET
PUBLIC DEFINE_CLUSTER
PUBLIC DISTRIBUTION_MAP
PUBLIC ADD2BIT16

AWAIT_IND EQU 0             ;*** THESE ARE THE IDENTIFICATION
ADVANCE_IND EQU 1           ;*** CODES RECOGNIZED BY THE
CREATE_EVC_IND EQU 2        ;*** GATEKEEPER IN LEVEL II OF
CREATE_SEQ_IND EQU 3        ;*** MCORTEX
TICKET_IND EQU 4
READ_IND EQU 5
CREATE_PROC_IND EQU 6
PREEMPT_IND EQU 7
DEFINE_CLUSTER_IND EQU 8
DISTRIBUTION_MAP_IND EQU 9

IF GATEMOD
GATEKEEPER_IP DW 0036H
```

```
GATEKEEPER_CS DW 0BADH
ELSE
GATEKEEPER_IP DW 0068H          ;#### 1 #### <---------------
GATEKEEPER_CS DW 0B4CH          ;#### 2 #### <---------------
ENDIF
GATEKEEPER EQU DWORD PTR GATEKEEPER_IP

CSEG

;*** AWAIT *** AWAIT *** AWAIT *** AWAIT *** AWAIT *********/

AWAIT:

PUSH ES
MOV SI,2[BX]               ;SI <-- PNT TO COUNT AWAITED
MOV BX,[BX]                ;BX <-- PNT TO NAME OF EVENT
MOV AL,AWAIT_IND
PUSH AX                    ;N <-- AWAIT INDICATOR
MOV AL,[BX]
PUSH AX                    ;BYT <-- NAME OF EVENT
MOV AX,[SI]                ;AX <-- COUNT AWAITED
PUSH AX                    ;WORDS <-- COUNT AWAITED
PUSH AX                    ;PTR_SEG <-- UNUSED WORD
PUSH AX                    ;PTR_OFFSET <--UNUSED WORD
CALLF GATEKEEPER
POP ES

RET

;*** ADVANCE *** ADVANCE *** ADVANCE *** ADVANCE **********/

ADVANCE:

PUSH ES
MOV BX,[BX]                ;BX <-- PTR TO NAME OF EVENT
MOV AL,ADVANCE_IND
PUSH AX                    ;N <-- ADVANCE INDICATER
MOV AL,[BX]
PUSH AX                    ;BYT <-- NAME OF EVENT
PUSH AX                    ;WORDS <-- UNUSED WORD
PUSH AX                    ;PTR_SEG <-- UNUSED WORD
PUSH AX                    ;PTR_OFFSET <--UNUSED WORD
CALLF GATEKEEPER
POP ES

RET

;*** CREATE_EVC *** CREATE_EVC *** CREATE_EVC *************/

CREATE_EVC:

PUSH ES
```

```
MOV BX,[BX]             ;BX <-- PTR TO NAME OF EVENT
MOV AL,CREATE_EVC_IND
PUSH AX                 ;N <-- CREATE_EVC INDICATOR
MOV AL,[BX]
PUSH AX                 ;BYT <-- NAME OF EVENT
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH AX                 ;PTR_SEG <-- UNUSED WORD
PUSH AX                 ;PTR_OFFSET <--UNUSED WORD
CALLF GATEKEEPER
POP ES

RET

;*** CREATE_SEQ *** CREATE_SEQ *** CREATE_SEQ *************/

CREATE_SEQ:

PUSH ES
MOV BX,[BX]             ;BX <-- PTR TO NAME OF SEQ
MOV AL,CREATE_SEQ_IND
PUSH AX                 ;N <-- CREATE_SEQ INDICATER
MOV AL,[BX]
PUSH AX                 ;BYT <-- NAME OF SEQ
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH AX                 ;PTR_SEG <-- UNUSED WORD
PUSH AX                 ;PTR_OFFSET <--UNUSED WORD
CALLF GATEKEEPER
POP ES

RET

;*** TICKET *** TICKET *** TICKET *** TICKET *** TICKET ***/

TICKET:

PUSH ES
PUSH ES                 ;TICKET NUMBER DUMMY STORAGE
MOV CX,SP               ;POINTER TO TICKET NUMBER
MOV BX,[BX]             ;BX <-- PTR TO TICKET NAME
MOV AL,TICKET_IND
PUSH AX                 ;N <-- TICKET INDICATER
MOV AL,[BX]
PUSH AX                 ;BYT <-- TICKET NAME
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH SS                 ;PTR_SEG <-- TICKET NUMBER SEG
PUSH CX                 ;PTR_OFFSET <-- TICKET NUMBER POINTER
CALLF GATEKEEPER
POP BX                  ;RETRIEVE TICKET NUMBER
POP ES

RET
```

```
;*** READ *** READ *** READ *** READ *** READ *** READ ****/

READ:

PUSH ES
PUSH ES                  ;EVENT COUNT DUMMY STORAGE
MOV CX,SP                ;POINTER TO EVENT COUNT
MOV BX,[BX]              ;BX <-- PTR TO EVENT NAME
MOV AL,READ_IND
PUSH AX                   ;N <-- READ INDICATER
MOV AL,[BX]
PUSH AX                   ;BYT <-- EVANT NAME
PUSH AX                   ;BYT <-- UNUSED WORD
PUSH SS                   ;PTR_SEG <-- EVENT COUNT SEGMENT
PUSH CX                   ;PTR_OFFSET <-- EVENT COUNT POINTER
CALLF GATEKEEPER
POP BX                    ;RETRIEVE EVENT COUNT
POP ES

RET

;*** CREATE_PROC *** CREATE_PROC *** CREATE_PROC **********/

CREATE_PROC:

PUSH ES
MOV SI,14[BX]             ;SI <-- PTR TO PROCESS ES
PUSH WORD PTR [SI]        ;STACK PROCESS ES
MOV SI,12[BX]             ;SI <-- PTR TO PROCESS DS
PUSH WORD PTR [SI]        ;STACK PROCESS DS
MOV SI, 10[BX]            ;SI <-- PTR TO PROCESS CS
PUSH WORD PTR [SI]        ;STACK PROCESS CS
MOV SI, 8[BX]             ;SI <-- PTR TO PROCESS IP
PUSH WORD PTR [SI]        ;STACK PROCESS IP
MOV SI, 6[BX]             ;SI <-- PTR TO PROCESS SS
PUSH WORD PTR [SI]        ;STACK PROCESS SS
MOV SI, 4[BX]             ;SI <-- PTR TO PROCESS SP
PUSH WORD PTR [SI]        ;STACK PROCESS SP
MOV SI,2[BX]              ;SI <-- PTR TO PROCESS PRIORITY
MOV AH,[SI]               ;GET PROCESS PRIORITY
MOV SI,[BX]               ;SI <-- PTR TO PROCESS ID
MOV AL,[SI]               ;GET PROCESS ID
PUSH AX                   ;STACK PROCESS PRIORITY AND ID
MOV CX,SP                 ;POINTER TO DATA
MOV AL,CREATE_PROC_IND
PUSH AX                   ;N <-- CREATE PROCESS IND
PUSH AX                   ;BYT <-- UNUSED WORD
PUSH AX                   ;WORDS <-- UNUSED WORD
PUSH SS                   ;PROC_PTR SEGMENT <-- STACK SEG
PUSH CX                   ;PROC_PTR OFFSET <-- DATA POINTER
CALLF GATEKEEPER
ADD SP,14                 ;REMOVE STACKED DATA
```

```
POP ES

RET

;*** PREEMPT *** PREEMPT *** PREEMPT *** PREEMPT **********/

PREEMPT:

PUSH ES
MOV BX,[BX]             ;BX <-- PTR TO NAME OF PROCESS
MOV AL,PREEMPT_IND
PUSH AX                 ;N <-- PREEMPT INDICATER
MOV AL,[BX]
PUSH AX                 ;BYTE <-- PREEMPT PROCESS NAME
PUSH AX                 ;WORDS <-- UNUSED WORD
PUSH AX                 ;PTR_SEG <-- UNUSED WORD
PUSH AX                 ;PTR_OFFSET <-- UNUSED WORD
CALLF GATEKEEPER
POP ES

RET

;***    DEFINE_CLUSTER ***    DEFINE_CLUSTER ***         **/

DEFINE_CLUSTER:

PUSH ES
MOV  BX, [BX]           ;BX <-- PTR TO LOCAL$CLUSTER$ADDR
MOV  AL, DEFINE_CLUSTER_IND
PUSH AX                 ;N <-- DEFINE_CLUSTER_IND
PUSH AX                 ;BYT <-- UNUSED WORD
PUSH WORD PTR [BX]      ;WORDS <-- LOCAL$CLUSTER$ADDR
PUSH AX                 ;PTR_SEG <-- UNUSED WORD
PUSH AX                 ;PTR_OFFSET <-- UNUSED WORD
CALLF GATEKEEPER
POP  ES
RET

;***   DISTRIBUTION_MAP   ***   DISTRIBUTON_MAP   ***     **/


DISTRIBUTION_MAP:

PUSH ES
MOV  SI, 4[BX]          ;SI <-- PTR TO GROUP ADDRESS
PUSH WORD PTR [SI]      ;STACK THE GROUP ADDRESS
MOV  SI, 2[BX]          ;SI <-- PTR TO ID OF MAP_TYPE
MOV  AH, [SI]
MOV  SI, [BX]           ;SI <-- PTR TO MAP_TYPE
MOV  AL, [SI]           ;AL <-- MAP_TYPE
PUSH AX                 ;STACK ID AND MAP_TYPE
MOV  CX, SP             ;POINTER TO DATA
MOV  AL, DISTRIBUTION_MAP_IND
PUSH AX                    ;N <-- DISTRIB_MAP_IND
PUSH AX                    ;BYT <-- UNUSED WORD
PUSH AX                    ;WORD <-- UNUSED WORD
PUSH SS                    ;MAP_PTR_SEG <-- SS
PUSH CX                    ;MAP_PTR_OFFSET <-- DATA PTR
CALLF GATEKEEPER
ADD  SP, 4
POP  ES
RET



;*** ADD2BIT16 *** ADD2BIT16 *** ADD2BIT16 *** ADD2BIT16 **/

ADD2BIT16:

MOV SI,[BX]                ;SI <-- PTR TO BIT(16)#1
MOV BX,2[BX]               ;BX <-- PTR TO BIT(16)#2
MOV BX,[BX]                ;BX <-- BIT(16)#2
ADD BX,[SI]                ;BX <-- BIT(16)#1 + BIT(16)#2

RET

END
```

# APPENDIX E

## DEMONSTRATION PROGRAM SOURCE CODE

The model of processes that demonstrates the distributivity of MCORTEX over Ethernet is illustrated in Figure 11. The interactions that are occurring at each cluster are as follows.

Cluster 1 - The MSLREACT (missile launcher reaction) process and TRACKER (target tracking) process get a ticket through the kernel (SYSTEM$IO) to write to the Ethernet Request Block. This corresponds to user transparent simultaneous requests for Ethernet access. The NI3010 Driver and Packet Processor is processing Ethernet Request Packets and Ethernet packets. All processes are competing for access to GLOBAL data via the kernel.

Cluster 2 - the MSLORDER (generates missile orders) and TRKRPRT (track reporting) processes are multiplexed on one real processor and are scheduled and blocked based on the interaction with the Cluster 1 processes. The NI3010 Driver and Packet Processor performs the same function as that in Cluster 1. The code is identical with the exception of the initialization module. Recall that this module in each cluster is responsible for the creation of eventcounts and sequencers, as well as calls to DEFINE$CLUSTER and DISTRIBUTION$MAP, which are cluster and eventcount-dependent (distribution of) procedures.

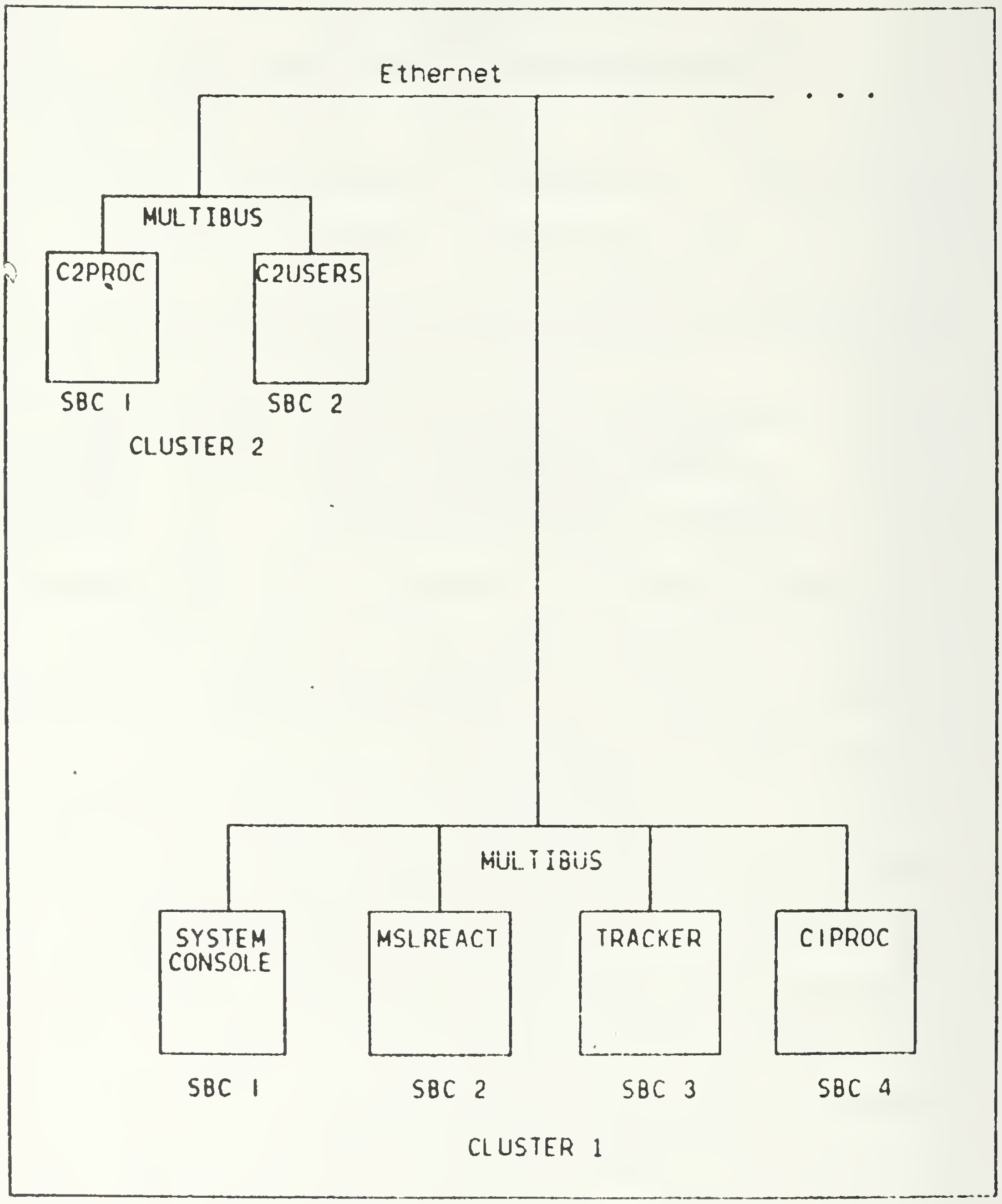


FIGURE 11 - Demonstration Model

Both Clusters - there is no actual computations being done by any MCORTEX process, so Ethernet Request Packets and Ethernet Packets are being generated at the fastest possible rate. Any possible timing problems would be exposed by this demonstration process. None were noted, and the processes performed as expected.

The system console shown in Figure 11 is used to monitor (using DDT86) changes in GLOBAL data and shared memory structures. A process that automatically provides diagnostic and display support is under development for RTC STAR. This process will execute under CP/M-86 on single board computer 1 in each cluster. Source code for the demonstration model, except for Link86 input option files and the NI3010 Driver, follows. The input option files and NI3010 Driver are contained in Appendices F and K respectively.

```
*************************************************************
*  MSLTINIT is the initialization module for the process   *
*  that  simulates the training of a missile launcher as   *
*  a result of orders received from the MSLORDER  process *
*  of Cluster 2. This module is linked as shown in         *
*  MSLREACT.INP or LAUNCH/T.INP in Appendix F.             *
*                                                          *
*    PL/I-86 Source File Name : MSLTINIT.PLI               *
*                                                          *
*************************************************************

msltinit:       procedure options (main);

        %include 'sysdef.pli';


        /* begin */

           call create_proc ('02'b4, 'fc'b4,
                             '0600'b4, '04d8'b4, '0023'b4,
                             '0439'b4, '04d8'b4, '04d8'b4);
           call await ('fe'b4, '01'b4);

end msltinit;

*************************************************************
* MSLTRAIN is the main module of a process that "responds"*
* to commands   issued by MSLORDER. It is a consumer of    *
* missile orders. It signals its use of a command   by     *
* advancing distributed eventcount MISSILE_ORDER_OUT.      *
* It is linked as shown in MSLREACT.INP or LAUNCH/T.INP    *
* of Appendix F.                                           *
*                                                          *
*   PL/I-86 Source File Name     : MSLTRAIN.PLI            *
*   MCORTEX Command Module Name  : MSLREACT.CMD            *
*   MXTRACE Command Module Name  : LAUNCH/T.CMD            *
*                                                          *
*************************************************************

msltrain:       procedure ;

                %replace

                        infinity        by 32767,
                        one             by '0001'b4;

                %include 'sysdef.pli';
                DECLARE

                        i fixed bin (15),
                        k bit (16) static init ('0000'b4);

                /* end DECLARATIONS */

        /*  main */

        do i = 0 to infinity;

                k = add2bit16(k, one);
                call await (MISSILE_ORDER_IN, k);


                /* consume() */

                call advance (MISSILE_ORDER_OUT);


        end;  /* do i */

end msltrain;

*************************************************************
* TRKDINIT is the initialization module for the process     *
* TRACKER (CMD filename). It is linked as shown in          *
* TRACKER.INP or TRKER/T.INP of Appendix F.                 *
*                                                           *
*     PL/I-86 Source File Name : TRKDINIT.PLI               *
*                                                           *
*************************************************************

trkdinit:       procedure options (main);

        %include 'sysdef.pli';



        /* begin */

        call create_proc ('01'b4, 'fc'b4,
                          '0900'b4, '06ff'b4, '0023'b4,
                          '0439'b4, '06ff'b4, '06ff'b4);
        call await ('fe'b4, '01'b4);

end trkdinit;
```

```
*****************************************************************
* TRKDETECT is  the main module of a producing   process       *
* that  simulates the detection of tracks (air contacts)       *
* and advances eventcount TRACK_IN to signal that the          *
* next iteration of track data is available. The consumer      *
* process   is TRKRPRT, located at Cluster 2. This module      *
* is linked as shown in TRACKER.INP or TRKER/T.INP of          *
* Appendix F.                                                  *
*                                                              *
*   MCORTEX Command Module Name : TRACKER.CMD                  *
*   MXTRACE Command Module Name : TRKER/T.CMD                  *
*                                                              *
*****************************************************************

trkdetect:      procedure ;

                %replace

                        FOREVER          by '1'b,
                        one              by '0001'b4,
                        buffer_length    by  50;

                %include 'sysdef.pli';

                DECLARE

                        i fixed bin (15),
                        (k,buffer_ub,buffer_lb) bit (16);

                /* end DECLARATIONS */

        /*  main */

        do i = 0 to 32000;

        /* simulation of track input data*/

        /*    Input from real-time sensor here    */

                call advance (TRACK_IN);
                buffer_ub = read(TRACK_IN);
                put skip(2) edit  ('Eventcount value = ',
                                   buffer_ub)(a,b4(5));
                buffer_lb = read (TRACK_OUT);
                put skip(2) edit ('Eventcount value = ',
                                   buffer_lb)(a,b4(5));
                if ((binary(buffer_ub)-
                   binary(buffer_lb))>=buffer_length) then
                do;
                       k = add2bit16(buffer_lb.one);
                       call await (TRACK_OUT, k);
                end;
        end;  /* do FOREVER */
end trkdetect;

****************************************************************
* C2UINIT is the initialization module for the Cluster 2      *
* processes that are multiplexed on SBC 2. This module is     *
* linked as shown in C2USERS.INP or C2USER/T.INP in           *
* Appendix F.                                                  *
*                                                              *
*     PL/I-86 Source File Name : C2UINIT.PLI                   *
*                                                              *
****************************************************************

c2_users_init:  procedure options (main);

        %include 'sysdef.pli';



        /* begin */

                     /* missile order */
         call create_proc ('03'b4, 'fc'b4,
                           '0820'b4, '06ff'b4, '0029'b4,
                           '0439'b4, '06ff'b4, '06ff'b4);

         call create_proc ('04'b4, 'fc'b4,
                           '0940'b4, '06ff'b4, '00de'b4,
                           '0439'b4, '06ff'b4, '06ff'b4);
         call await ('fe'b4, '01'b4);

end c2_users_init;
```

```
***************************************************************
* MSLORDER is the  main module of a producing process at      *
* Cluster 2 that  simulates issuing missile orders. It        *
* signals the next iteration of missile orders by             *
* advancing eventcount MISSILE_ORDER_IN. The consumer is      *
* MSLREACT at cluster 1. This  module is linked as shown      *
* in C2USERS.INP or C2USER/T.INP in Appendix F.               *
*                                                             *
*     PL/I-86 Source File Name : MSLORDER.PLI                 *
*     Contained in :                                          *
*                                                             *
*        MCORTEX Command Module Name : C2USERS.CMD            *
*        MXTRACE Command Module Name : C2USER/T.CMD           *
*                                                             *
***************************************************************

mslorder:        procedure ;

                 %replace

                         infinity         by 32767,
                         one              by '0001'b4,
                         buffer_length    by  50;

                 %include 'sysdef.pli';

                 DECLARE

                         i fixed bin (15),
                         (k,buffer_ub,buffer_lb) bit (16);

                 /* end DECLARATIONS */

        /*  main */

        do i = 0 to infinity;

        /* simulation of missile order */


                 call advance (MISSILE_ORDER_IN);
                 buffer_ub = read(MISSILE_ORDER_IN);
                 put skip(2) edit  ('Eventcount value = ',
                                    buffer_ub)(a,b4(5));
                 buffer_lb = read (MISSILE_ORDER_OUT);
                 put skip(2) edit ('Eventcount value = ',
                                    buffer_lb)(a,b4(5));
```

```
                if ((binary(buffer_ub)-
                  binary(buffer_lb))>=buffer_length) then
                do;
                        k = add2bit16(buffer_lb,one);
                        call await (MISSILE_ORDER_OUT, k);
                end;

        end;  /* do i */

end mslorder;

************************************************************
* TRKRPRT is the main module of a process at  Cluster 2    *
* that  simulates the consumption of track detection data. *
* It signals its consumption by advancing eventcount       *
* TRACK_OUT.  This module is linked as shown in            *
* C2USERS.INP or  C2USER/T.INP of Appendix F.              *
*                                                          *
*     PL/I-86 Source File Name : TRKRPRT.PLI               *
*                                                          *
************************************************************

trkrprt:        procedure ;

                %replace

                        infinity        by 32767,
                        one             by '0001'b4;

                %include 'sysdef.pli';

                DECLARE

                        i fixed bin (15),
                        k bit (16) static init ('0000'b4);

                /* end DECLARATIONS */

        /*  main */

        do i = 0 to infinity;

                k = add2bit16(k, one);
                call await (TRACK_IN, k);

                /* consume() */

                call advance (TRACK_OUT);
        end;  /* do i */
end trkrprt;
```

# APPENDIX F

## LINK86 Input Option Files

The INPUT option directs LINK86 to obtain further command line input from an indicated file. This reduces the amount of interactive typing needed to link various modules together. In essence, the input file is a batch file scanned by LINK86. For example, the modules shown in C1PROC.INP are linked with the command : LINK86 C1PROC[I], where I denotes that C1PROC.INP contain the actual files to be linked. The name appearing on the lefthand side of the equal sign in the LINK86 option files is the name assigned to the CMD module. Therefore, LINK86 C1PROC[I] produces the CMD module C1PROC.CMD. Details concerning this procedure may be found in [Ref. 18].

```
**************************************************************
**************************************************************
***            MCORTEX input option file                   ***
**************************************************************
**************************************************************

MCORTEX = TEX/TRC [code[ab[B2Ø]],data[ab[B4Ø]]]

**************************************************************
**************************************************************
***            C1PROC  input option file                   ***
**************************************************************
**************************************************************

c1proc =
sysinit1 [code[ab[439]],data[ab[8ØØ],m[Ø],ad[82]],map[all]],
sysdev,
asmrout,
gatemod

**************************************************************
**************************************************************
***            TRACKER input option file                   ***
**************************************************************
**************************************************************

tracker =
trkdinit [code[ab[439]],data[ab[6ff],m[Ø],ad[82]],map[all]],
trkdetec,
gatemod

**************************************************************
**************************************************************
***            MSLREACT input option file                  ***
**************************************************************
**************************************************************

mslreact =
msltinit [code[ab[439]],data[ab[4d8],m[Ø],ad[82]],map[all]],
msltrain,
gatemod
```

```
*************************************************************
*************************************************************
***            C2PROC  input  option  file                ***
*************************************************************
*************************************************************


c2proc =
sysinit2 [code[ab[439]],data[ab[8ØØ],m[Ø],ad[82]],map[all]],
sysdev,
asmrout,
gatemod

*************************************************************
*************************************************************
***            C2USERS input  option  file                ***
*************************************************************
*************************************************************


c2users =
c2uinit [code[ab[439]],data[ab[6ff],m[Ø],ad[82]],map[all]],
mslorder,
trkrprt,
gatemod

*************************************************************
*************************************************************
***            MXTRACE input option file                  ***
*************************************************************
*************************************************************

MXTRACE = TRX/TRC [code[ab[A6C]],data[ab[A8C]]]

*************************************************************
*************************************************************
***            C1PROC/T input  option file                ***
*************************************************************
*************************************************************


c1proc/t =
sysinit1 [code[ab[439]],data[ab[8ØØ],m[Ø],ad[82]],map[all]],
sysdev,
asmrout,
gatetrc
```

```
***************************************************************
***************************************************************
***               TRKER/T input  option  file               ***
***************************************************************
***************************************************************

trker/t =
trkdinit [code[ab[439]],data[ab[6ff],m[0],ad[82]],map[all]],
trkdetec,
gatetrc

***************************************************************
***************************************************************
***               LAUNCH/T input  option  file              ***
***************************************************************
***************************************************************

launch/t =
msltinit [code[ab[439]],DATA[AB[4d8],M[0],AD[82]],map[all]],
msltrain,
gatetrc

***************************************************************
***************************************************************
***               C2PROC/T input  option  file              ***
***************************************************************
***************************************************************

c2proc/t =
sysinit2 [code[ab[439]],data[ab[800],m[0],ad[82]],map[all]],
sysdev,
asmrout,
gatetrc

***************************************************************
***************************************************************
***               C2USER/T input  option  file              ***
***************************************************************
***************************************************************

c2user/t =
c2uinit [code[ab[439]],data[ab[6ff],m[0],ad[82]],map[all]],
mslorder,
trkrprt,
gatetrc
```

# APPENDIX G

## LEVEL II MCORTEX SOURCE CODE

The LEVEL II source code, written in PL/M-86, is contained in file LEVEL2.SRC. Due to the conditional compilation switches contained in the code, it should be compiled for either the MCORTEX version or MXTRACE version. Files are provided to be used with the SUBMIT utility [Ref. 19]. The MCORTEX version of LEVEL II is compiled by using the SUBMIT file L2CMPM.CSD (LEVEL2 compile, MCORTEX). LEVEL II is one of the relocatable code modules shown in the SUBMIT file LNKKM.CSD, which is used to link the modules together for KORE.OPS. After linking, the resultant file must be located using the LOC86 utility. This is provided for in the SUBMIT file LOCKM.CSD (locate MCORTEX). The file KORE is created and becomes KORE.OPS after transfer to the multi-user CP/M-86 system. KORE.OPS is loaded by MCORTEX.CMD under the CP/M-86 operating system. Memory maps for KORE.OPS and KORE.TRC are provided at the end of Appendix H. The map information comes from KORE.MP2 after compiling, linking, and locating the applicable files.

In the source listing for LEVEL2.SRC, the executable code must begin in column 7 (see L2CMPM.CSD). It appears left justified in this listing due to thesis format requirements.

```
************************************************************
************************************************************
***            SUBMIT file L2CMPM.CSD                    ***
************************************************************
************************************************************

:F1:PLM86 :F1:LEVEL2.SRC SET(MCORTEX) NOCOND LEFTMARGIN(7)
                         LARGE

************************************************************
************************************************************
***            SUBMIT file L2CMPT.CSD                    ***
************************************************************
************************************************************

:F1:PLM86 :F1:LEVEL2.SRC RESET(MCORTEX) NOCOND LEFTMARGIN(7)
                         LARGE
```

```
/*************************************************************/
/*************************************************************/
/*0009*********************************************************/
     /* FILE:          LEVEL2.SRC
        VERSION:       BREWER 8-18-84
        PROCEDURES
         DEFINED:     GATE$KEEPER            CREATE$EVC
                      READ                   AWAIT
                      ADVANCE                PREEMT
                      TICKET                 CREATE$PROC
                      DEFINE$CLUSTER         DISTRIBUTION$MAP
                      OUT$CHAR               OUT$LINE
                      OUT$NUM                OUT$DNUM
                      SEND$CHAR              OUT$HEX
                      RECV$CHAR              IN$CHAR
                      IN$NUM                 IN$DNUM
                      IN$HEX

   REMARKS:  !!! CAUTION !!! !!! CAUTION !!! !!! CAUTION!!!
             IF NEW USER SERVICES ARE ADDED TO THIS MODULE
             OR CHANGES ARE MADE TO EXISTING ONES, MAKE
             SURE THE LOCATOR MAP (FILE: KORE.MP2) IS CHECK-
             ED TO SEE IF THE LOCATION OF 'GATE$KEEPER' HAS
             NOT CHANGED.  THE ABSOLUTE ADDRESS OF THIS
             PROCEDURE HAS BEEN SUPPLIED TO THE GATE$MODULE
             IN FILE: GATE.SRC.  IF IT HAS CHANGED THE NEW
             ADDRESS SHOULD BE UPDATED IN FILE:  GATE.SRC
             AND RECOMPILED.  ALL USER PROCESSES WILL HAVE
             TO BE RELINKED WITH FILE:  GATE.OBJ AND
             RELOCATED.

             LITERAL DECLARATIONS GIVEN AT THE BEGINNING
             OF SEVERAL MODULES ARE LOCAL TO THE ENTIRE
             MODULE.  HOWEVER, SOME ARE LISTED THE SAME
             IN MORE THAN ONE MODULE.  THE VALUE AND
             THEREFORE THE MEANING OF THE LITERAL IS
             COMMUNICATED ACROSS MODULE BOUNDARIES.
             'NOT$FOUND' USED IN LOCATE$EVC AND
             CREATE$EVC IS AN EXAMPLE.  TO CHANGE IT IN
             ONE MODULE AND NOT THE OTHER WOULD KILL
             THE CREATION OF ANY NEW EVENTCOUNTS BY THE
             OS.


             CONDITIONAL COMPILATION COMMANDS ARE USED TO
             PRODUCE TWO VERSIONS OF THE MCORTEX OPERATING
             SYSTEM. "MCORTEX" IS THE VERSION WITHOUT ANY
             I/O PERTAINING TO ENTRY OF OS PRIMITIVES. WITH
             THIS VERSION IT IS EXPECTED THAT THE USER HAS
             COMPLETED DEBUGGING OF USER PROCESS CODE AND
             THIS IS NO LONGER NECESSARY. IN CONTRAST,
             THE CODE BRACKETED BY 'NOT MCORTEX' IS THE CODE
```

```
              FOR THE TRACE VERSION OF MCORTEX KNOWN AS
              "MXTRACE." THIS VERSION PROVIDES DIAGNOSTIC
              'HOOKS' INTO THE OS AND SHOULD BE USED DURING
              THE CODE DEVELOPMENT STAGES.
                                                                 */
/*****************************************************************/




/*0073***********************************************************/

     L2$MODULE: DO;


/*****************************************************************/
/*****************************************************************/
/* LOCAL DECLARATIONS                                            */

DECLARE
   MAX$CPU                   LITERALLY     '10',
   MAX$VPS$CPU               LITERALLY     '10',
   MAX$CPU$$$$MAX$VPS$CPU    LITERALLY    '100',
   FALSE                     LITERALLY      '0',
   READY                     LITERALLY      '1',
   RUNNING                   LITERALLY      '3',
   WAITING                   LITERALLY      '7',
   TRUE                      LITERALLY    '119',
   NOT$FOUND                 LITERALLY    '255',
   PORT$CA                   LITERALLY  '00CAH',
   RESET                     LITERALLY      '0',
   ENET                      LITERALLY      '0',
   ERB$BLOCK$LENGTH          LITERALLY     '20',
   EVC$TYPE                  LITERALLY      '0',
   ERB$READ                  LITERALLY   '0FCH',
   ERB$WRITE                 LITERALLY   '0FDH',
   ERB$WRITE$REQUEST         LITERALLY   '0FFH',
   INT$RETURN                LITERALLY    '77H';

/*0102***********************************************************/
/* PROCESSOR DATA SEGMENT TABLE                                  */
/*    DELARED PUBLIC IN MODULE 'L1$MODULE'                       */
/*                     IN FILE   'LEVEL1   '                     */

DECLARE PRDS STRUCTURE
   (CPU$NUMBER                BYTE,
    VP$START                  BYTE,
    VP$END                    BYTE,
    VPS$PER$CPU             BYTE,
    LAST$RUN                BYTE,
    COUNTER                 WORD)           EXTERNAL;

/*0115**************************************************************/
/* GLOBAL DATA BASE DECLARATIONS                                   */
/*    DECLARED PUBLIC IN FILE  'GLOBAL.SRC'                        */
/*                 IN MODULE'GLOBAL$MODULE'                        */


DECLARE VPM( MAX$CPU$$$$MAX$VPS$CPU ) STRUCTURE
     (VP$ID                  BYTE,
      STATE                  BYTE,
      VP$PRIORITY            BYTE,
      EVC$THREAD             BYTE,
      EVC$AW$VALUE           WORD,
      SP$REG                 WORD,
      SS$REG                 WORD)           EXTERNAL;

DECLARE
   LOCAL$CLUSTER$ADDR  .  WORD             EXTERNAL;


DECLARE
   EVENTS                 BYTE             EXTERNAL;

DECLARE EVC$TBL (100) STRUCTURE
   (EVC$NAME               BYTE,
    VALUE                  WORD,
    REMOTE$ADDR            WORD,
    THREAD                 BYTE)            EXTERNAL;

DECLARE
   SEQUENCERS             BYTE             EXTERNAL;

DECLARE SEQ$TABLE (100) STRUCTURE
   (SEQ$NAME               BYTE,
    SEQ$VALUE              WORD)            EXTERNAL;

DECLARE
   NR$VPS( MAX$CPU )      BYTE             EXTERNAL,
   NR$RPS                 BYTE             EXTERNAL,
   HDW$INT$FLAG (MAX$CPU )BYTE             EXTERNAL,
   GLOBAL$LOCK            BYTE             EXTERNAL;

/*0156**************************************************************/
/* DECLARATION OF EXTERNAL PROCEDURE REFERENCES                    */
/*    DECLARED PUBLIC IN FILE  'LEVEL1.SRC'                        */
/*                 IN MODULE 'LEVEL1$MODULE'                       */

VPSCHEDULER:  PROCEDURE EXTERNAL; END;
       /* IN FILE 'SCHED.ASM' */
```

```
RET$VP     :   PROCEDURE BYTE EXTERNAL; END;


LOCATE$EVC :   PROCEDURE (EVENT$NAME) BYTE EXTERNAL;
   DECLARE EVENT$NAME BYTE;
END;

LOCATE$SEQ :   PROCEDURE (SEQ$NAME) BYTE EXTERNAL;
   DECLARE SEQ$NAME BYTE;
END;

/*0175**********************************************************/
/*  DIAGNOSTIC MESSAGES OR "HOOKS"                             */


$IF NOT MCORTEX


   DECLARE
   MSG16(*) BYTE INITIAL('ENTERING  PREEMPT',13,10,'%'),
   MSG17(*) BYTE INITIAL('ISSUING   INTERRUPT!!',13,10,'%'),
   MSG18(*) BYTE INITIAL('ENTERING  AWAIT',10,13,'%'),
   MSG19(*) BYTE INITIAL('ENTERING  ADVANCE ',10,13,'%'),
   MSG21(*) BYTE INITIAL('ENTERING  CREATE$EVC FOR %'),
   MSG23(*) BYTE INITIAL('ENTERING  READ FOR EVC:  %'),
   MSG24(*) BYTE INITIAL('ENTERING  TICKET',13,10,'%'),
   MSG25(*) BYTE INITIAL('ENTERING  CREATE$SEQ  %'),
   MSG26(*) BYTE INITIAL('ENTERING  CREATE$PROC',10,13,'%'),
   MSG27(*) BYTE INITIAL(10,'ENTERING  GATE$KEEPER  N= %');

DECLARE
   CR LITERALLY  '0DH',
   LF LITERALLY  '0AH';


$ENDIF

/*0201**********************************************************/
/***************************************************************/
/**  GATE$KEEPER   PROCEDURE              BREWER 8-18-81 ****/
/***************************************************************/
/*  THIS PROCEDURE IS THE ENTRY INTO THE OPERATING            */
/*  SYSTEM DOMAIN FROM THE USER DOMAIN.  THIS IS THE          */
/*  ACCESS POINT TO THE UTILITY/SERVICE ROUTINES AVAIL-       */
/*  ABLE TO THE USER. THIS PROCEDURE IS CALLED BY THE         */
/*  GATE MODULE WHICH IS LINKED WITH THE USER PROGRAM.        */
/*  IT IS THE GATE MODULE WHICH PROVIDES TRANSLATION          */
/*  FROM THE USER DESIRED FUNCTION TO THE FORMAT REQUIR-      */
/*  ED FOR THE GATEKEEPER.  THE GATEKEEPER CALLS THE          */
/*  DESIRED UTILITY/SERVICE PROCEDURE IN LEVEL2 OF THE        */
/*  OPERATING SYSTEM AGAIN PERFORMING THE NECESSARY           */
/*  TRANSLATION FOR A PROPER CALL.  THE TRANSLATIONS ARE  */
/*  INVISIBLE TO THE USER.  THE GATEKEEPER ADDRESS IS     */
/*  PROVIDED TO THE GATE MODULE TO BE USED FOR THE IN-    */
/*  DIRECT CALL.                                          */
/*                                                        */
/*  THE PARAMETER LIST IS PROVIDED FOR CONVENIENCE AND    */
/*  REPRESENTS NO FIXED MEANING, EXCEPT FOR 'N'.          */
/*     N      FUNCTION CODE PROVIDED BY GATE              */
/*     BYT    BYTE VARIABLE FOR TRANSLATION               */
/*     WORDS  WORD      "      "        "                 */
/*     PTR    POINTER VARIABLE FOR TRANSLATION            */
/*0243*****************************************************/

GATE$KEEPER: PROCEDURE(N, BYT, WORDS, PTR) REENTRANT PUBLIC;

DECLARE
   (N, BYT) BYTE,
    WORDS WORD,
    PTR POINTER;
/* I-O SERVICES ARE NOT ACKNOWLEDGED FOR TWO REASONS:      */
/*    1.  THEY ARE CALLED SO OFTEN THAT DIAGNOSTIC OUTPUT  */
/*        WOULD BE TOO CLUTTERED.                          */
/*    2.  THEY THEMSELVES PRODUCES I-O EFFECTS THAT        */
/*        ACKNOWLEDGE THEY ARE BEING CALLED.               */

$IF NOT MCORTEX

   IF N < 10 THEN DO;
      CALL OUT$LINE(@MSG27);
      CALL OUT$NUM(N);
      CALL OUT$CHAR(CR);
      CALL OUT$CHAR(LF);
   END;

$ENDIF

   DO CASE N;                          /*  N  */
      CALL AWAIT(BYT,WORDS);           /*  0  */
      CALL ADVANCE(BYT);               /*  1  */
      CALL CREATE$EVC(BYT);            /*  2  */
      CALL CREATE$SEQ(BYT);            /*  3  */
      CALL TICKET(BYT,PTR);            /*  4  */
      CALL READ(BYT,PTR);              /*  5  */
      CALL CREATE$PROC(PTR);           /*  6  */
      CALL PREEMPT( BYT );             /*  7  */
      CALL DEFINE$CLUSTER(WORDS);      /*  8  */
      CALL DISTRIBUTION$MAP(PTR);      /*  9  */


$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
```

```
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

      CALL OUT$CHAR(BYT);                  /* 10  */
      CALL OUT$LINE(PTR);                  /* 11  */
      CALL OUT$NUM(BYT);                   /* 12  */
      CALL OUT$DNUM(WORDS);                /* 13  */
      CALL IN$CHAR(PTR);                   /* 14  */
      CALL IN$NUM(PTR);                    /* 15  */
      CALL IN$DNUM(PTR);                   /* 16  */

$ENDIF

   END;   /* CASE */
      RETURN;
END;   /* GATE$KEEPER */

/*0283**********************************************************/
/*  CREATE$EVC  PROCEDURE                       BREWER 8-18-84 */
/*-------------------------------------------------------------*/
/* CREATES EVENTCOUNT FOR INTER-PROCESS SYNCHRONIZATION.       */
/* EVENTCOUNT IS INITIALIZED TO 0 IN THE EVENTCOUNT TABLE.*/
/***************************************************************/
CREATE$EVC: PROCEDURE(NAME) REENTRANT PUBLIC;

   DECLARE NAME BYTE;

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
   CALL OUT$LINE(@MSG21);
   CALL OUT$NUM(NAME);
   CALL OUT$CHAR(CR);
   CALL OUT$CHAR(LF);

$ENDIF

   /* ASSERT GLOBAL LOCK */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119); END;

      IF /* THE EVENTCOUNT DOES NOT ALREADY EXIST */
        LOCATE$EVC(NAME) = NOT$FOUND THEN DO;
          /* CREATE THE EVENTCOUNT ENTRY BY ADDING THE  */
          /* NEW EVENTCOUNT TO THE END OF THE EVC$TABLE */
          EVC$TBL(EVENTS).EVC$NAME = NAME;
          EVC$TBL(EVENTS).VALUE = 0;
          EVC$TBL(EVENTS).REMOTE$ADDR = LOCAL$CLUSTER$ADDR;
          EVC$TBL(EVENTS).THREAD = 255;
          /* INCREMENT THE SIZE OF THE EVC$TABLE */
          EVENTS = EVENTS + 1;
         END; /* CREATE THE EVENTCOUNT */
         /* RELEASE THE GLOBAL LOCK */
        GLOBAL$LOCK = 0;
        RETURN;
     END; /* CREATE$EVC PROCEDURE */


/*0324*********************************************************/
/*   READ PROCEDURE                            BREWER 8-18-84  */
/*-------------------------------------------------------------*/
/* THIS PROCEDURE ALLOWS USERS TO READ THE PRESENT VALUE       */
/* OF THE SPECIFIED EVENT$COUNT WITHOUT MAKING ANY             */
/* CHANGES.  A POINTER IS PASSED TO PROVIDE A BASE TO A        */
/* VARIABLE IN THE CALLING ROUTINE FOR PASSING THE RETURN      */
/* VALUE BACK TO THE CALLING ROUTINE.                          */
/***************************************************************/

READ:  PROCEDURE( EVC$NAME, RETS$PTR ) REENTRANT PUBLIC;

     DECLARE
        EVC$NAME              BYTE,
        EVCTBL$INDEX          BYTE,
        RETS$PTR              POINTER,
        EVC$VALUE$RET         BASED RETS$PTR WORD;

        /*  SET THE GLOBAL LOCK  */
        DO WHILE LOCKSET(@GLOBAL$LOCK,119);  END;

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
   CALL OUT$LINE(@MSG23);
   CALL OUT$NUM(EVC$NAME);
   CALL OUT$CHAR(CR);
   CALL OUT$CHAR(LF);

$ENDIF

        /*  OBTAIN INDEX  */
   EVCTBL$INDEX = LOCATE$EVC( EVC$NAME );

        /*  OBTAIN VALUE  */
   EVC$VALUE$RET = EVC$TBL( EVCTBL$INDEX ).VALUE;

       /*  UNLOCK GLOBAL LOCK  */
   GLOBAL$LOCK = 0 ;
   RETURN;
END;   /*  READ PROCEDURE  */


/*0368*********************************************************/
/*   AWAIT PROCEDURE                                           */
/*-------------------------------------------------------------*/
```

```
/* INTER PROCESS SYNCHRONIZATION PRIMITIVE.  SUSPENDS      */
/* EXECUTION OF RUNNING PROCESS UNTIL THE EVENTCOUNT HAS   */
/* REACHED THE SPECIFIED THRESHOLD VALUE, "AWAITED$VALUE."*/
/* USED BY THE OPERATING SYSTEM FOR THE MANAGEMENT OF      */
/* SYSTEM RESOURCES.                                       */
/*********************************************************/

AWAIT: PROCEDURE(EVC$ID,AWAITED$VALUE) REENTRANT PUBLIC;

   DECLARE
      AWAITED$VALUE     WORD,
      (EVC$ID, NEED$SCHED, RUNNING$VP,EVCTBL$INDEX) BYTE;

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   CALL OUT$LINE(@MSG18);

$ENDIF

   /* LOCK GLOBAL LOCK */
   DO WHILE LOCK$SET(@GLOBAL$LOCK, 119);  END;
   NEED$SCHED = TRUE;

   /*  DETERMINE THE RUNNING VIRTUAL PROCESSOR  */
   RUNNING$VP = RET$VP;

   /*  GET EVC INDEX  */
   EVCTBL$INDEX = LOCATE$EVC(EVC$ID);
   /* DETERMINE IF CURRENT VALUE IS LESS THAN THE
      AWAITED VALUE */
   IF EVC$TBL(EVCTBL$INDEX).VALUE < AWAITED$VALUE THEN DO;
      /* BLOCK PROCESS */
    VPM(RUNNING$VP).EVC$THREAD=EVC$TBL(EVCTBL$INDEX).THREAD;
    VPM(RUNNING$VP).EVC$AW$VALUE = AWAITED$VALUE;
    EVC$TBL( EVCTBL$INDEX ).THREAD = RUNNING$VP;
    DISABLE;
    PRDS.LAST$RUN = RUNNING$VP;
    VPM(RUNNING$VP).STATE = WAITING;
   END;     /* BLOCK PROCESS */

   ELSE       /* DO NOT BLOCK PROCESS */
    NEED$SCHED = FALSE;

    /*  SCHEDULE THE VIRTUAL PROCESSOR  */
    IF NEED$SCHED = TRUE  THEN
      CALL VPSCHEDULER;              /* NO RETURN */

      /* UNLOCK GLOBAL LOCK */
      GLOBAL$LOCK = 0;
      RETURN;
     END;   /* AWAIT PROCEDURE */


/*0427***********************************************************/
/*    ADVANCE   PROCEDURE                        PREWER 8-18-84   */
/*--------------------------------------------------------------*/
/*  INTER PROCESS SYNCHRONIZATION PRIMITIVE.  INDICATES         */
/*  SPECIFIED EVENT HAS OCCURED BY ADVANCING(INCREMENTING)*/
/*  THE ASSOCIATED EVENTCOUNT.  EVENT IS BROADCAST TO ALL */
/*  VIRTUAL PROCESSORS AWAITING THAT EVENT.                    */
/*  A CALL TO ADVANCE WILL RESULT IN A CALL TO THE SCHED- */
/*  ULER, EVEN IF THE ADVANCING OF THE EVENTCOUNT DOES      */
/*  RESULT IN AWAKENING ANY NEW PROCESSES. THUS, ANY         */
/*  HIGHER PRIORITY ONBOARD PROCESS READIED BY AN OFF-      */
/*  BOARD OPERATION WOULD BE SCHEDULED NEXT.                  */
/*--------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$LINE                                    */
/*                  SYSTEM$IO                                   */
/*                  VPSCHEDULER (NO RETURN)                     */
/***************************************************************/

ADVANCE: PROCEDURE(EVC$ID) REENTRANT PUBLIC;

   DECLARE
      (EVC$ID, EVCTBL$INDEX      )                        BYTE,
      (SAVE, RUNNING$VP, DUMMY$VAR, I)                    BYTE,
      CLUSTER$ADDR                                        WORD;

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   CALL OUT$LINE(@MSG19);

$ENDIF

   /* LOCK THE GLOBAL LOCK */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119);  END;

    RUNNING$VP = RET$VP;
    EVCTBL$INDEX = LOCATE$EVC(EVC$ID);
    EVC$TBL(EVCTBL$INDEX).VALUE=EVC$TBL(EVCTBL$INDEX).VALUE + 1;
    IF EVC$TBL(EVCTBL$INDEX).REMOTE$ADDR <> LOCAL$CLUSTER$ADDR
    THEN DO;

         /* REMOTE COPY IS NEEDED - THE CONVENTION IS:

            AN EVENTCOUNT THAT HAS A REMOTE COPY WILL
            NOT HAVE ITS REMOTE$ADDR FIELD EQUAL TO THE
            LOCAL$CLUSTER$ADDR.
```

```
                                                      */

          CLUSTER$ADDR=EVC$TBL(EVCTBLINDEX).REMOTE$ADDR
                        XOR LOCAL$CLUSTER$ADDR;
          GLOBAL$LOCK = 0;
          CALL SYSTEM$IO(ENET,EVC$TYPE,EVC$ID,
               EVC$TBL(EVCTBL$INDEX).VALUE,CLUSTER$ADDR);
          DO WHILE LOCK$SET(@GLOBAL$LOCK,119); END;

        END;     /* ITD */
        SAVE = 255;
        I = EVC$TBL( EVCTBL$INDEX ).THREAD;
        DO WHILE  I <> 255;
         IF VPM(I).EVC$AW$VALUE <= EVC$TBL(EVCTBL$INDEX).VALUE
         THEN DO;       /* AWAKEN THE PROCESS */
          VPM(I).STATE = READY;

          VPM(I).EVC$AW$VALUE = 0;
          IF SAVE = 255 THEN DO; /*THIS FIRST ONE IN LIST*/
            DUMMY$VAR = VPM(I).EVC$THREAD;
            EVC$TBL(EVCTBL$INDEX).THREAD = DUMMY$VAR;
            VPM( I ).EVC$THREAD = 255;
            I = EVC$TBL( EVCTBL$INDEX ).THREAD;
          END;  /* IF FIRST */
          ELSE DO;       /* THEN THIS NOT FIRST IN LIST */
            VPM( SAVE ).EVC$THREAD = VPM( I ).EVC$THREAD;
            VPM( I ).EVC$THREAD = 255;
            I = VPM( SAVE ).EVC$THREAD;
          END;  /* IF NOT FIRST */
         END;     /* IF AWAKEN */
         ELSE DO;       /* DO NOT AWAKEN THIS PROCESS */
          SAVE = I;
          I = VPM( I ).EVC$THREAD;
         END;      /* IF NOT AWAKEN */
    END;      /* DO WHILE */
      PRDS.LAST$RUN = RUNNING$VP;
      VPM(RUNNING$VP).STATE = READY;
      CALL VPSCHEDULER;        /* NO RETURN */
      /* UNLOCK THE GLOBAL LOCK */
      GLOBAL$LOCK = 0;
      RETURN;
END;  /* ADVANCE  PROCEDURE */

/*0518*****************************************************/
/*  PREEMPT  PROCEDURE                      BREWER 8-18-84  */
/*-------------------------------------------------------*/
/* THIS PROCEDURE AWAKENS A HI PRIOITY PROCESS LEAVING    */
/* THE CURRENT RUNNING PROCESS IN THE  READY STATE AND    */
/* CALLS FOR A RESCHEDULING.  THE HIGH PRIORITY PROCESS   */
/* SHOULD BLOCK ITSELF WHEN FINISHED.                     */
/*   IF THE VP$ID IS 'FE' OR THE MONITOR PROCESS, IT WILL */
/* MAKE IT READY WHERE-EVER IT IS IN THE VPM.  THE FOLLOW-*/
/* ING CODE DOES NOT TAKE ADVANTAGE OF THE FACT THAT        */
/* CURRENTLY IT IS THE THIRD ENTRY IN THE VPM FOR EACH      */
/* REAL PROCESOR.                                           */
/*----------------------------------------------------------*/
/* CALLS MADE TO:  OUTLINE, VPSCHEDULER                     */
/************************************************************/

PREEMPT:  PROCEDURE( VP$ID ) REENTRANT PUBLIC;

   DECLARE (VP$ID,SEARCH$ST,SEARCH$END,CPU,INDEX) BYTE;

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   CALL OUT$LINE( @MSG16 );

$ENDIF

   IF VP$ID <> 0FEH THEN DO; /* NORMAL PREEMT */
      /* SEARCH VPM FOR INDEX FOR ID  */
      SEARCH$ST = 0;
      DO CPU = 0 TO (NR$RPS - 1);
        SEARCH$END = SEARCH$ST + NR$VPS( CPU ) - 1 ;
        DO INDEX = SEARCH$ST TO SEARCH$END;
          IF VPM( INDEX ).VP$ID = VP$ID  THEN GO TO FOUND;
        END; /* DO INDEX */
        SEARCH$ST = SEARCH$ST + MAX$VPS$CPU;
      END; /* DO CPU */
           /* CASE IF NOT FOUND IS NOT ACCOUNTED FOR CURRENTLY *
      FOUND:
         /* LOCK THE GLOBAL LOCK */
         DO WHILE LOCK$SET(@GLOBAL$LOCK,119);  END;
           /* SET PREEMPTED VP TO READY */
           VPM( INDEX ).STATE = READY;
           /* NEED HARDWARE INTR OR RE-SCHED  */
           IF ( CPU = PRDS.CPU$NUMBER ) THEN DO;
              INDEX = RET$VP;   /* DETERMINE RUNNING PROCESS */
              DISABLE;
              PRDS.LAST$RUN = INDEX;
              VPM( INDEX ).STATE = READY;  /* SET TO READY */
              CALL VPSCHEDULER;   /* NO RETURN */
           END;
           ELSE DO;      /* CAUSE HARDWARE INTERRUPT */

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

           CALL OUT$LINE(@MSG17);
```

```
$ENDIF
             HDW$INT$FLAG( CPU ) = TRUE;
             DISABLE;  OUTPUT( PORT$CA ) =  80H;
             CALL TIME(1);
             OUTPUT  PORT$CA ) = RESET;  ENABLE;
           END;
   END; /* NORMAL PREEMT */
   ELSE DO; /* PREEMT THE MONITOR */
     /* SEARCH VPM FOR ALL ID´S OF 0FEH */
     SEARCH$ST  = 0;
     DO WHILE LOCK$SET(@GLOBAL$LOCK,119); END;
     DO CPU = 0 TO (NR$RPS - 1);
       SEARCH$END = SEARCH$ST + NR$VPS( CPU ) - 1;
       /* SET ALL INT$FLAGS EXCEPT THIS CPU´S */
       IF PRDS.CPU$NUMBER <> CPU THEN
          HDW$INT$FLAG( CPU ) = TRUE;
          DO INDEX = SEARCH$ST TO SEARCH$END;
            IF VPM( INDEX ).VP$ID = VP$ID THEN
               VPM( INDEX ).STATE = READY;
          END; /* DO */
          SEARCH$ST = SEARCH$ST + MAX$VPS$CPU;
     END; /* ALL MONITOR PROCESS SET TO READY */
      /* INTERRUPT THE OTHER CPU´S AND
         RESCHEDULE THIS ONE               */

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   CALL OUT$LINE(@MSG17);

$ENDIF

      DISABLE;
      OUTPUT( PORT$CA ) = 80H;
      CALL TIME(1);
      OUTPUT( PORT$CA ) = RESET;
      ENABLE;
      INDEX = RET$VP;
      DISABLE;
      PRDS.LAST$RUN = INDEX;
      VPM(INDEX).STATE = READY;
      CALL VPSCHEDULER;  /* NO RETURN */
   END; /* ELSE
   /* UNLOCK GLOBAL MEMORY */
   GLOBAL$LOCK = 0;
   RETURN;
END;   /*  PREEMPT PROCEDURE  */
```

```
/*0631*********************************************************/
/*    CREATE$SEQ  PROCEDURE                    BREWER 8-18-84    */
/*-------------------------------------------------------------*/
/* CREATOR OF INTER PROCESS SEQUENCER PRIMITIVES FOR USER */
/* PROGRAMS.  CREATES A SPECIFIED SEQUENCER AND INITIAL-  */
/* IZES IT TO 0, BY ADDING THE SEQUENCER TO THE END OF THE*/
/* SEQUENCER TABLE.                                        */
/*-------------------------------------------------------------*/
/* CALLS MADE TO:  OUT$LINE               OUT$CHAR         */
/*                 OUT$HEX                                 */
/*********************************************************/

CREATE$SEQ: PROCEDURE(NAME) REENTRANT PUBLIC;

   DECLARE NAME BYTE;

   /* ASSERT GLOBAL LOCK */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119); END;

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   CALL OUT$LINE(@MSG25);
   CALL OUT$HEX(NAME);
   CALL OUT$CHAR(CR);
   CALL OUT$CHAR(LF);

$ENDIF


   IF /* THE SEQUENCER DOES NOT ALREADY EXIST, IE */
     LOCATE$SEQ(NAME) = NOT$FOUND THEN DO;
       /* CREATE THE SEQUENCER ENTRY BY ADDING THE  */
       /* NEW SEQUENCER TO THE END OF THE SEQ$TABLE */
     SEQ$TABLE(SEQUENCERS).SEQ$NAME   = NAME;
     SEQ$TABLE(SEQUENCERS).SEQ$VALUE  =  0;
       /*  INCREMENT NUMBER OF SEQUENCERS  */
     SEQUENCERS = SEQUENCERS + 1;
   END; /* CREATE THE SEQUENCER */
     /* RELEASE THE GLOBAL LOCK */
   GLOBAL$LOCK = 0;
   RETURN;
END; /* CREATE$SEQ PROCEDURE */


/*0678*********************************************************/
/*  TICKET   PROCEDURE                         BREWER 8-18-84    */
/*-------------------------------------------------------------*/
/* INTER-VIRTUAL PROCESSOR SEQUENCER RPIMITIVE FOR  USER   */
/* PROGRAM.  SIMILAR TO 'TAKE A NUMBER AND WAIT.'  RETURNS*/
/* PRESENT VALUE OF SPECIFIED SEQUENCER AND INCREMENTS THE*/
/* SEQUENCER.  A POINTER IS PASSED TO PROVIDE A BASE TO A */
/* VARIABLE IN THE CALLING ROUTINE FOR PASSING THE RETURN */
/* VALUE BACK TO THE CALLING ROUTINE.                     */
/*------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$LINE                              */
/**********************************************************/

TICKET:  PROCEDURE( SEQ$NAME, RETS$PTR ) REENTRANT PUBLIC;


   DECLARE
      SEQ$NAME        BYTE,
      SEQTBL$INDEX    BYTE,
      RETS$PTR        POINTER,
      SEQ$VALUE$RET   BASED RETS$PTR WORD;

      /* ASSERT GLOBAL LOCK  */
      DO WHILE LOCKSET(@GLOBAL$LOCK,119);  END;

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   CALL OUT$LINE(@MSG24);


$ENDIF

   /*  OBTAIN SEQ$NAME INDEX  */
   SEQTBL$INDEX = LOCATE$SEQ( SEQ$NAME );
   /*  OBTAIN SEQUENCER VALUE  */
   SEQ$VALUE$RET = SEQ$TABLE( SEQTBL$INDEX ).SEQ$VALUE;
   /*  INCREMENT SEQUENCER  */
   SEQ$TABLE( SEQTBL$INDEX ).SEQ$VALUE  =
                SEQ$TABLE(SEQTBL$INDEX).SEQ$VALUE  +  1 ;

   /*  UNLOCK THE GLOBAL LOCK  */
   GLOBAL$LOCK  =  0 ;
   RETURN;
END;    /*  TICKET PROCEDURE  */


/*0727****************************************************/
/*          CREATE$PROC    PROCEDURE   BREWER 8-18-84     */
/*------------------------------------------------------*/
/*  THIS PROCEDURE CREATES A PROCESS FOR THE USER AS      */
/*  SPECIFIED BY THE INPUT PARAMETERS CONTAINED IN A      */
/*  STRUCTURE IN THE GATE MODULE.  THE PARAMETER PASSED   */
/*  IS A POINTER WHICH POINTS TO THIS STRUCTURE.          */
/*  INFO CONTAINED IN THIS STRUCTURE IS:  PROCESS ID,     */
/*  PROCESS PRIORITY, THE DESIRED PROC STACK LOCATION,      */
/*  AND THE PROCESS CODE STARTING LOCATION WHICH IS         */
/*  IS TWO ELEMENTS: THE IP REGISTER (OFFSET) AND THE       */
/*  CS REGISTER (CODE SEGMENT).                             */
/*----------------------------------------------------------*/
/* CALLS MADE TO:   OUTLINE                                 */
/************************************************************/

CREATE$PROC: PROCEDURE( PROC$PTR ) REENTRANT PUBLIC;

   DECLARE
      PROC$PTR        POINTER,
      PROC$TABLE BASED PROC$PTR STRUCTURE
      (PROC$ID              BYTE,
       PROC$PRI             BYTE,
       PROC$SP              WORD,
       PROC$SS              WORD,
       PROC$IP              WORD,
       PROC$CS              WORD,
       PROC$DS              WORD,
       PROC$ES              WORD);


   DECLARE
      (PS1, PS2)   WORD,
       TEMP        BYTE;

   DECLARE PROC$STACK$PTR POINTER AT(@PS1),
      PROC$STACK BASED PROC$STACK$PTR STRUCTURE
      (LENGTH(0FEH)       BYTE,
       RET$TYPE           WORD,
       BP                 WORD,
       DI                 WORD,
       SI                 WORD,
       DS                 WORD,
       DX                 WORD,
       CX                 WORD,
       AX                 WORD,
       BX                 WORD,
       ES                 WORD,
       IP                 WORD,
       CS                 WORD,
       FL                 WORD);

$IF NOT MCORTEX

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

   CALL OUT$LINE(@MSG26);

$ENDIF
```

```
   /*  TO SET UP PROC$STACK$PTR  */
   PS1 = PROC$TABLE.PROC$SP - 118H;
   PS2 = PROC$TABLE.PROC$SS;

   PROC$STACK.RET$TYPE = INT$RETURN;
   PROC$STACK.BP = PROC$TABLE.PROC$SP;
   PROC$STACK.DI = 0;
   PROC$STACK.SI = 0;
   PROC$STACK.DS = PROC$TABLE.PROC$DS;
   PROC$STACK.DX = 0;
   PROC$STACK.CX = 0;
   PROC$STACK.AX = 0;
   PROC$STACK.BX = 0;
   PROC$STACK.ES = PROC$TABLE.PROC$ES;
   PROC$STACK.IP = PROC$TABLE.PROC$IP;
   PROC$STACK.CS = PROC$TABLE.PROC$CS;
   PROC$STACK.FL = 200H;  /*SET IF FLAG (ENABLE INTR)*/

   /* SET GLOBAL LOCK */
   DO WHILE LOCKSET(@GLOBAL$LOCK,119);  END;

   IF PRDS.VPS$PER$CPU < MAX$VPS$CPU  THEN DO;
      TEMP = PRDS.VPS$PER$CPU + PRDS.VP$START;
      VPM( TEMP ).VP$ID = PROC$TABLE.PROC$ID;
      VPM( TEMP ).STATE = 01;       /* READY */
      VPM( TEMP ).VP$PRIORITY = PROC$TABLE.PROC$PRI;
      VPM( TEMP ).EVC$THREAD = 255;
      VPM( TEMP ).EVC$AW$VALUE = 0;
      VPM( TEMP ).SP$REG = PROC$TABLE.PROC$SP - 1AH;
      VPM( TEMP ).SS$REG = PROC$TABLE.PROC$SS;

      PRDS.VPS$PER$CPU = PRDS.VPS$PER$CPU + 1;
      PRDS.VP$END = PRDS.VP$END + 1;
      NR$VPS( PRDS.CPU$NUMBER ) =
      NR$VPS(PRDS.CPU$NUMBER) + 1;
   END;   /* DO */

   /* RELEASE THE GLOBAL LOCK */
   GLOBAL$LOCK = 0;
   RETURN;
END;        /*  CREATE$PROCESS  */


/*0832*********************************************************/
/*   SYSTEM$IO        PROCEDURE        BREWER 8-18-84      */
/*-----------------------------------------------------------*/
/* PROCESSES A REQUEST FROM THE ADVANCE PROCEDURE (AND     */
/* OTHERS TO BE DEVELOPED) TO ADVANCE THE VALUE OF AN      */
/* EVENTCOUNT THAT HAS A REMOTE COPY. ALTHOUGH THE         */
/* CURRENT IMPLEMENTATION IS LIMITED TO THE ETHERNET AS    */
/* THE MEDIUM FOR DISTRIBUTED EVENTCOUNTS, THE PROCEDURE   */
/* IS WRITTEN TO ALLOW FOR THE EXTENSION TO OTHER DATA    */
/* COMMUNICATION MEDIA.                                   */
/*                                                        */
/* FUNCTIONALITY:                                         */
/*     QUEUES UP REQUESTS IN AN ETHERNET REQUEST BLOCK    */
/*     (ERB) FOR CONSUMPTION BY THE ETHERNET COMMUNICATION*/
/*     CONTROLLER BOARD (ECCB) DEVICE HANDLER.            */
/*--------------------------------------------------------*/
/* CALLS MADE TO: READ                                    */
/*                ADVANCE                                 */
/*                TICKET                                  */
/**********************************************************/


SYSTEM$IO: PROCEDURE (PATH,REQUEST$TYPE,NAME,VALUE,ADDR)
                 PUBLIC REENTRANT;

   DECLARE
      (PATH, REQUEST$TYPE, NAME, ERB$INDEX, INDEX)       BYTE,
      (VALUE, ADDR, I, J           )                      WORD;


   DECLARE
      ERB(ERB$BLOCK$LENGTH)  STRUCTURE
         ( COMMAND           BYTE,
           TYPE$NAME         BYTE,
           NAME$VALUE        WORD,
           REMOTE$ADDR       WORD) AT (10000H);

   IF PATH = ENET THEN
   DO;

      DO CASE REQUEST$TYPE;

         DO;         /* IT'S ETHERNET AND EVENTCOUNT  */
           CALL TICKET(ERB$WRITE$REQUEST, @I);
              /* I NOW HAS THE VALUE OF TICKET RETURNED */
           CALL READ(ERB$WRITE, @J);
              /* J NOW HAS THE VALUE OF ERB$WRITE */
           DO WHILE (J < I);
              CALL TIME (10);
              /* 1 MS DELAY ==> REDUCE BUS CONTENTION */
              CALL READ(ERB$WRITE, @J);
           END;    /* DO WHILE */
              /* WRITE TO ERB, IF IT'S NOT FULL */
           CALL READ(ERB$READ, @J);
           DO WHILE ( (I-J) >= ERB$BLOCK$LENGTH);
              /* IT'S FULL SO DO A "BUSY WAIT" */
              CALL TIME (60);
              /* DELAY ONE PACKET TRANSMISSION TIME
                 QUANTUM */
              CALL READ(ERB$READ, @J);
```

```
            END;

            /* SLOT OPEN SO WRITE TO ERB */
            ERB$INDEX = I MOD ERB$BLOCK$LENGTH;
            ERB (ERB$INDEX).COMMAND = REQUEST$TYPE;
            ERB (ERB$INDEX).TYPE$NAME = NAME;
            ERB (ERB$INDEX).NAME$VALUE = VALUE;
            ERB (ERB$INDEX).REMOTE$ADDR = ADDR;
            /* NEED TO ADVANCE THE VALUE OF ERB$WRITE */

            DO WHILE LOCK$SET(@GLOBAL$LOCK, 119);
               /* ASSERT LOCK */
            END;
            INDEX = LOCATE$EVC(ERB$WRITE);
            EVC$TBL(INDEX).VALUE = EVC$TBL(INDEX).VALUE + 1;
            GLOBAL$LOCK = 0; /* RELEASE */

            /* NOTE THAT THIS AVOIDS THE UNNECESSARY OVER- */
            /*    HEAD OF THE ADVANCE PROCEDURE */
         END;  /* DO BLOCK */

         DO;   /* STUB FOR NOW  */
         END;

      END;  /* REQUEST$TYPE */


   END;       /* PATH */

END;        /* SYSTEM$IO */


/*0923*****************************************************/
/*    DEFINE$CLUSTER PROCEDURE               BREWER 8-18-84  */
/*-------------------------------------------------------*/
/* THIS PROCEDURE IS CALLED ONLY ONE TIME AT EACH CLUSTER.*/
/* ITS SOLE PURPOSE IS TO DEFINE THE LOCAL$CLUSTER ADDRESS.*/
/* THIS PROCEDURE CALL MUST BE THE FIRST CALL IN THE INIT  */
/* PROCESS BROUGHT UP IN EACH CLUSTER.                     */
/*                                                         */
/*********************************************************/

DEFINE$CLUSTER: PROCEDURE (CLUSTER$ID) REENTRANT PUBLIC;

   DECLARE CLUSTER$ID                 WORD,
           I                          BYTE;

   LOCAL$CLUSTER$ADDR = CLUSTER$ID;
   /* FOR NOW OTHER ENTITIES FIELDS ARE UNINITIALIZED */

   EVC$TBL(0).REMOTE$ADDR=CLUSTER$ID;
   /* FIRST ENTRY IN TABLE IS A RESERVED SYSTEM EVENTCOUNT */
```

```
END;   /* DEFINE$CLUSTER */



/*0947**********************************************************/
/*    DISTRIBUTION$MAP PROCEDURE              BREWER 8-18-84   */
/*-------------------------------------------------------------*/
/* THIS PROCEDURE ASSIGNS GROUP ADDRESSES TO THE               */
/* REMOTE$ADDR FIELD OF THE DISTRIBUTED ENTITY. THIS IS A */
/* SYSTEM MANAGEMENT DECISION - THE USER (ALTHOUGH SYSTEM)*/
/* PROCESSES DO NOT MAKE CALLS TO THIS PROCEDURE.              */
/*                                                             */
/***************************************************************/

DISTRIBUTION$MAP: PROCEDURE (MAP$PTR) REENTRANT PUBLIC;

   DECLARE
      MAP$PTR POINTER,
      TBL$INDEX BYTE,
      MAP$TABLE BASED MAP$PTR STRUCTURE
         (MAP$TYPE              BYTE,
          ID                    BYTE,
          CLUSTER$ADDR          WORD);

   DO CASE MAP$TABLE.MAP$TYPE;

     DO;       /* EVENTCOUNT TYPE */
      TBL$INDEX = LOCATE$EVC (MAP$TABLE.ID);
      EVC$TBL(TBL$INDEX).REMOTE$ADDR=MAP$TABLE.CLUSTER$ADDR;
     END;

     DO;
      /* STUB */
     END;

   END;   /* DO CASE */

END;      /* DISTRIBUTION$MAP */


$IF NOT MCORTEX

/* CONDITIONAL COMPILATION OF PROCEDURES
ASSOCIATED WITH    *** MXTRACE ***             */

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

/*0990**********************************************************/
/*    IN$CHAR    PROCEDURE                    BREWER 8-18-84   */
/*-------------------------------------------------------------*/
/* GETS A CHAR FROM THE SERIAL PORT.  CHAR IS !!!NOT!!!    */
/* ECHOED.  THAT IS RESPONSIBILTY OF USER IN THIS CASE.    */
/* INPUT TO SERIAL PORT VIA SBC861 DOWN LOAD PROGRAM MAY   */
/* NOT BE ACCEPTED.                                         */
/* POINTER IS PROVIDED BY USER SO HE CAN BE RETURNED THE   */
/* CHARACTER .                                              */
/*-- -------- ---------------------------------------------*/
/* CALLS MADE TO:  RECV$CAHR                                */
/**********************************************************/

IN$CHAR:  PROCEDURE ( RET$PTR ) REENTRANT PUBLIC;

            DECLARE
               RET$PTR POINTER,
               INCHR BASED RET$PTR BYTE;

            DISABLE;
            INCHR = RECV$CHAR;
            ENABLE;


            RETURN;
        END; /* IN$CHAR */

/*1017******************************************************/
/*   IN$NUM      PROCEDURE             BREWER  8-18-84      */
/*---------------------------------------------------------*/
/*  GETS TWO ASCII CHAR FROM THE SERIAL PORT, EXAMINES     */
/*  THEM TO SEE IF THEY ARE IN THE SET Ø..F HEX AND FORMS */
/*  A BYTE VALUE.  EACH VALID HEX DIGIT IS ECHOED TO THE   */
/*  CRT.  IMPROPER CHAR ARE IGNORED.  NO ALLOWANCES ARE    */
/*  MADE FOR WRONG DIGITS.  GET IT RIGHT THE FIRST TIME.   */
/*  IF YOU ARE INDIRECTLY ACCESSING THE SERIAL PORT VIA    */
/*  THE SBC861 DOWN LOAD PROGRAM FROM THE MDS SYSTEM       */
/*  INPUT MAY NOT BE ACCEPTED.  A POINTER IS PASSED BY THE*/
/*  USER SO THAT HE RETURNED THE CHARACTER.                */
/*---------------------------------------------------------*/
/*  CALLS MADE TO:  IN$HEX                                  */
/**********************************************************/

         IN$NUM:  PROCEDURE ( RET$PTR ) REENTRANT PUBLIC;
            DECLARE
               RET$PTR     POINTER,
               NUM BASED RET$PTR BYTE;

            DISABLE;
            NUM = IN$HEX;
            ENABLE;
            RETURN;
        END; /* IN$NUM */


/*1045******************************************************/
/**********************************************************/
```

```
/*   OUT$CHAR     PROCEDURE                     BREWER 8-18-84  */
/*--------------------------------------------------------------*/
/* SENDS A BYTE TO THE SERIAL PORT                              */
/*--------------------------------------------------------------*/
/* CALL MADE TO:   SEND$CHAR                                    */
/****************************************************************/

        OUT$CHAR: PROCEDURE( CHAR ) REENTRANT PUBLIC;

           DECLARE CHAR BYTE;
           DISABLE;
           CALL SEND$CHAR( CHAR );
           ENABLE;
           RETURN;
        END;


/*1064**********************************************************/
/*   OUT$LINE PROCEDURE                    BREWER 8-18-84       */
/*--------------------------------------------------------------*/
/* USING A POINTER TO A BUFFER IT WILL OUTPUT AN ENTIRE         */
/* LINE THRU THE SERIAL PORT UNTIL AN '%' IS ENCOUNTERED        */
/* OR 80 CHARACTERS IS REACHED--WHICH EVER IS FIRST.  CR'S*/
/* AND LF'S CAN BE INCLUDED.                                    */
/*--------------------------------------------------------------*/
/* CALLS MADE TO:  SEND$CHAR                                    */
/****************************************************************/

        OUT$LINE: PROCEDURE( LINE$PTR ) REENTRANT PUBLIC;

           DECLARE
              LINE$PTR POINTER,
              LINE BASED LINE$PTR (80) BYTE,
              II BYTE;

           DISABLE;
           DO II = 0 TO 79;
              IF LINE( II ) = '%' THEN  GO TO DONE;
              CALL SEND$CHAR( LINE( II ) );
           END;
           DONE:   ENABLE;
           RETURN;
        END;


/*1092**********************************************************/
/*  OUT$NUM          PROCEDURE                 BREWER   8-18-84 */
/*--------------------------------------------------------------*/
/*  OUTPUTS A BYTE VAULE NUMBER THRU THE SERIAL PORT            */
/*--------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$HEX                                     */
/****************************************************************/
```

```
        OUT$NUM: PROCEDURE( NUM ) REENTRANT PUBLIC;

           DECLARE NUM BYTE;

           DISABLE;
           CALL OUT$HEX( NUM );
           ENABLE;
           RETURN;
        END;


/*1111**************************************************************/
/*  IN$DNUM        PROCEDURE                  BREWER  8-18-84    */
/*--------------------------------------------------------------*/
/* GETS FOUR ASCII FROM SERIAL PORT TO FORM WORD VALUE.        */
/* CRITERIA ARE THE SAME AS IN PROCEDURE IN$NUM.               */
/*--------------------------------------------------------------*/
/* CALLS MADE TO:  IN$HEX                                      */
/**************************************************************/

        IN$DNUM:  PROCEDURE ( RET$PTR ) REENTRANT PUBLIC;

           DECLARE
              RET$PTR     POINTER,
              DNUM BASED RET$PTR WORD,
             (H, L)   WORD;

           DISABLE;
           H = IN$HEX;
           H = SHL( H, 8 );
           L  = IN$HEX;
           DNUM = (H OR L);
           ENABLE;
           RETURN;
        END;


/*1137**************************************************************/
/*    OUT$DNUM     PROCEDURE                 BREWER 8-18-84      */
/*--------------------------------------------------------------*/
/*  OUTPUTS A WORD VALUE NUMBER VIA THE SERIAL PORT            */
/*--------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$HEX                                    */
/**************************************************************/

         OUT$DNUM: PROCEDURE( DNUM ) REENTRANT PUBLIC;

           DECLARE
              DNUM    WORD,
              SEND    BYTE;
```

```
            DISABLE;
            SEND = HIGH( DNUM );
            CALL OUT$HEX( SEND );
            SEND = LOW( DNUM );
            CALL OUT$HEX( SEND );
            ENABLE;
            RETURN;
         END;


/*1161*********************************************************/
/*   RECV$CHAR   PROCEDURE                  BREWER 8-18-84    */
/*------------------------------------------------------------*/
/* BOTTEM LEVEL PROCEDURE THAT OBTAINS A CHAR FROM THE        */
/* SERIAL PORT.  PARITY BIT IS REMOVED.  CHAR IS !!NOT!!      */
/* ECHOED.                                                    */
/*------------------------------------------------------------*/
/* CALLS MADE TO:    NONE                                     */
/**************************************************************/

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

RECV$CHAR:  PROCEDURE BYTE REENTRANT PUBLIC;

      DECLARE
         CHR      BYTE;

      /* CHECK PORT STATUS BIT 2 FOR RECEIVE-READY SIGNAL */
      DO WHILE (INPUT(ØDAH) AND Ø2H) = Ø;  END;
         CHR = (INPUT(ØD8H) AND Ø7FH);
         RETURN CHR;
END;


/*1185*********************************************************/
/*  SEND$CHAR     PROCEDURE              BREWER     8-18-84  */
/*------------------------------------------------------------*/
/*  OUTPUTS A BYTE THRU THE SERIAL PORT.  THIS IS NOT A       */
/*  SERVICE AVAILABLE THRU THE GATEKEEPER BUT IT IS CALLED*/
/*  BY MANY OF THOSE PROCEDURES.  IT WILL STOP SENDING        */
/*  (AND EVERYTHING ELSE) IF IT SEES A ^S AT INPUT.  ^Q       */
/*  WILL RELEASE THE PROCEDURE TO CONTINUE.                   */
/*  THE USER BEWARE!!!!! THIS IS ONLY A DIAGNOSTIC TOOL       */
/*  TO FREEZE THE CRT FOR STUDY.  RELEASING IT DOESN'T        */
/*  ASSURE NORMAL RESUMPTION OF EXECUTION.  (YOU MAY FORCE*/
/*  ALL BOARDS TO IDLE FOR EXAMPLE.)                          */
/*------------------------------------------------------------*/
/*  CALLS MADE TO:                                            */
/**************************************************************/

         SEND$CHAR: PROCEDURE(CHAR) REENTRANT PUBLIC;
         DECLARE (CHAR,INCHR) BYTE;

         /* CHECK PORT STATUS */
         INCHR = (INPUT(ØD8H) AND Ø7FH);
         IF INCHR = 13H THEN
            DO WHILE (INCHR <> 11H);
               IF ((INPUT(ØDAH) AND Ø2H) <> Ø) THEN
                  INCHR = (INPUT(ØD8H) AND Ø7FH);
            END;
         DO WHILE (INPUT(ØDAH) AND Ø1H) = Ø;   END;
         OUTPUT(ØD8H) = CHAR;
         RETURN;
      END;


/*1218*****************************************************/
/*  IN$HEX    PROCEDURE                    BREWER 8-18-84  */
/*---------------------------------------------------------*/
/*  GETS 2 HEX CHAR FROM THE SERIAL PORT AND IGNORES ANY-  */
/*  THING ELSE.  EACH VALID HEX DIGIT IS ECHOED TO THE     */
/*  SERIAL PORT.  A BYTE VALUE IS FORMED FROM THE TWO HEX  */
/*  CHAR.                                                  */
/*---------------------------------------------------------*/
/*  CALLS MADE TO:  RECV$CHAR                              */
/**********************************************************/

IN$HEX:  PROCEDURE  BYTE REENTRANT PUBLIC;

   DECLARE
      ASCII(*) BYTE DATA ('Ø123456789ABCDEF'),
      ASCIIL(*) BYTE DATA('Ø123456789',61H,62H,63H,64H,65H,
                          66H),

      (INCHR, HEXNUM, H, L)  BYTE,
      FOUND                  BYTE,
      STOP                   BYTE;

      /* GET HIGH PART OF BYTE */
      FOUND = Ø;
      DO WHILE NOT FOUND;
         /* IF INVALID CHAR IS INPUT, COME BACK HERE */
         INCHR = RECV$CHAR;
         H = Ø;
         STOP = Ø;
         /* COMPARE CHAR TO HEX CHAR SET */
         DO WHILE NOT STOP;
           IF (INCHR=ASCII(H)) OR (INCHR = ASCIIL(H)) THEN DO;
              STOP = ØFFH;
              FOUND = ØFFH;
              CALL SEND$CHAR( INCHR ); /* TO ECHO IT */
           END;
```

```
            ELSE DO;
               H = H + 1;
               IF H = 10H THEN STOP = 0FFH;
            END;  /* ELSE */
         END: /* DO WHILE */
         H = SHL( H, 4 );
      END; /* DO WHILE */
      FOUND = 0;
      DO WHILE NOT FOUND;
         INCHR = RECV$CHAR;
         L = 0H;
         STOP = 0;
         DO WHILE NOT STOP;
            IF (INCHR=ASCII(L)) OR (INCHR=ASCIIL(L)) THEN DO;
              STOP = 0FFH;
              FOUND = 0FFH;
              CALL SEND$CHAR(INCHR);
            END;
            ELSE DO;
              L = L + 1;
              IF L = 10H THEN STOP = 0FFH;
         END; /* ELSE */
      END;  /* DO WHILE */
   END;  /* DO WHILE */
   RETURN (H OR L);
END;  /* IN$HEX  */


/*1280***********************************************************/
/*     OUT$HEX    PROCEDURE                       BREWER 8-18-84 */
/*-------------------------------------------------------------*/
/*  TRANSLATES BYTE VALUES TO ASCII CHARACTERS AND OUTPUTS*/
/*  THEM THRU THE SERIAL PORT                                  */
/*-------------------------------------------------------------*/
/*  CALLS MADE TO:  SEND$CHAR                                  */
/***************************************************************/

         OUT$HEX: PROCEDURE(B) REENTRANT PUBLIC;

            DECLARE B BYTE;
            DECLARE ASCII(*) BYTE DATA ('0123456789ABCDEF');

            CALL SEND$CHAR(ASCII(SHR(B,4) AND 0FH));
            CALL SEND$CHAR(ASCII(B AND 0FH));
            RETURN;
         END;



/* END CONDITIONAL COMPILATION OF PROCEDURES NEEDED FOR
MXTRACE
```

```
                                                                    */
$ENDIF

/*********************************************************/

END;    /*  L2$MODULE  */

/*********************************************************/
/*********************************************************/
/*********************************************************/
```

# APPENDIX H

## LEVEL I MCORTEX SOURCE CODE

The LEVEL I source code, written in PL/M-86, is contained in file LEVEL1.SRC. The SUBMIT utility [Ref. 19] is used to compile either MCORTEX or MXTRACE versions of KORE. The MCORTEX version of LEVEL I is compiled by using the SUBMIT file L1CMPM.CSD. LEVEL I is one of the relocatable code modules shown in the SUBMIT file LNKKM.CSD in Appendix G. The SUBMIT file LOCKM.CSD is used to locate the various modules to file KORE. After transfer to the multi-user CP/M-86 system, the code is saved as KORE.OPS (as described in Appendix A). Analogous files are provided to generate KORE.TRC. The memory maps created by the linker and locator are included at the end of this appendix.

```
************************************************************
************************************************************
***                L1CMPM.CSD SUBMIT file                ***
************************************************************
************************************************************

:F1:PLM86 :F1:LEVEL1.SRC SET(MCORTEX) NOCOND LEFTMARGIN(7)
                         LARGE

************************************************************
************************************************************
***                L1CMPT.CSD SUBMIT file                ***
************************************************************
************************************************************

:F1:PLM86 :F1:LEVEL1.SRC RESET(MCORTEX) NOCOND LEFTMARGIN(7)
                         LARGE

************************************************************
************************************************************
***                LNKKM.CSD  SUBMIT file                ***
************************************************************
************************************************************

:F1:LINK86 :F1:LEVEL1.OBJ,:F1:LEVEL2.OBJ,:F1:SCHED.OBJ,&
:F1:INITK.OBJ,:F1:GLOBAL.OBJ TO :F1:KORE.LNK

************************************************************
************************************************************
***                LNKKT.CSD  SUBMIT file                ***
************************************************************
************************************************************

:F1:LINK86 :F1:LEVEL1.OBJ,:F1:LEVEL2.OBJ,:F1:SCHED.OBJ,&
:F1:INITK.OBJ,:F1:GLOBAL.OBJ TO :F1:KORE.LNK

************************************************************
************************************************************
***                LOCKM.CSD  SUBMIT file                ***
************************************************************
************************************************************

:F1:LOC86 :F1:KORE.LNK ADDRESSES(SEGMENTS(&
STACK(0C550H),&
INITMOD_CODE(04390H),&
GLOBALMODULE_DATA(0F5300H)))&
SEGSIZE(STACK(75H))&
RESERVE(0H TO 0B6FFH)


************************************************************
************************************************************
```

```
***                 LOCKT.CSD  SUBMIT file                        ***
**********************************************************************
**********************************************************************

:F1:LOC86 :F1:KORE.LNK ADDRESSES(SEGMENTS(&
STACK(0C5B0H),&
INITMOD_CODE(04390H),&
GLOBALMODULE_DATA(0E5300H)))&
SEGSIZE(STACK(75H))&
RESERVE(0H TO 0ABFFH)

**********************************************************************
**********************************************************************
***                   LEVEL1.SRC  file                            ***
**********************************************************************
**********************************************************************

/*0007**************************************************************/
/*******************************************************************/
/*******************************************************************/
/* FILE:            LEVEL1.SRC
   VERSION:         BREWER 8-18-84
   PROCEDURES
    DEFINED:    RET$VP          RDYTHISVP
                GETWORK         LOCATE$EVC
                LOCATE$SEQ      IDLE$PROC
                SAVE$CONTEXT    GET$SP
                MONITOR$PROC

   REMARKS:
   (1) WARNING: SEVERAL OF THE LITERAL DECLARATIONS BELOW
   HAVE A SIMILAR MEANING IN OTHER MODULES.  THAT MEAN-
   ING IS COMMUNICATED ACROSS MODULES BOUNDARIES.  BE
   CARFFUL WHEN CHANGING THEM.


   (2) CONDITIONAL COMPILATION FACILITIES ARE USED TO
   PRODUCE TWO OS VERSIONS. 'MCORTEX' PROVIDES NO
   DIAGNOSTIC ASSISTANCE, WHEREAS "MXTRACE" PROVIDES
   DISPLAY MESSAGES ANNOUNCING THE ENTRY INTO VARIOUS
   OS PRIMITIVES.
                                                                 */
/*******************************************************************/

L1$MODULE:  DO;

/*0036**************************************************************/
/*******************************************************************/
/*  LOCAL DECLARATIONS                                             */

DECLARE
   MAX$CPU                    LITERALLY     '10',
   MAX$VPS$CPU                  LITERALLY    '10',
   MAX$CPU$$$MAX$VPS$CPU        LITERALLY   '100',
   FALSE                        LITERALLY     '0',
   READY                        LITERALLY     '1',
   RUNNING                      LITERALLY     '3',
   WAITING                      LITERALLY     '7',
   TRUE                         LITERALLY   '119',
   NOT$FOUND                    LITERALLY   '255',
   PORT$C0                      LITERALLY '00C0H',
   PORT$C2                      LITERALLY '00C2H',
   PORT$CE                      LITERALLY '00CEH',
   PORT$CA                      LITERALLY '00CAH',
   RESET                        LITERALLY     '0',
   INT$RETURN                   LITERALLY   '77H',

$IF MCORTEX

/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
   IDLE$STACK$SEG               LITERALLY  '0C5DH',    /*********/
   IDLE$STACK$ABS               LITERALLY '0C5D0H',    /*********/
   INIT$STACK$SEG               LITERALLY  '0C65H',    /*********/
   INIT$STACK$ABS               LITERALLY '0C650H';    /*********/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/


$ELSE

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
   IDLE$STACK$SEG               LITERALLY  '0C63H',
   IDLE$STACK$ABS               LITERALLY '0C630H',
   INIT$STACK$SEG               LITERALLY  '0C6BH',
   INIT$STACK$ABS               LITERALLY '0C6B0H',
   MONITOR$STACK$SEG            LITERALLY  '0C73H',
   MONITOR$STACK$ABS            LITERALLY '0C730H';
   /**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
   /**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/


$ENDIF
```

```
/*0086*********************************************************/
/*  PROCESSOR DATA SEGMENT TABLE                               */
/*     INFORMATION RELEVANT TO THE PARTICULAR PHYSICAL         */
/*     PROCESSOR ON WHICH IT IS RESIDENT.                      */
/*                                                             */
/*     CPU$NUMBER:     UNIQUE SEQUENTIAL NUMBER ASSIGNED TO    */
/*                     THIS REAL PROCESSOR.                    */
/*     VP$START:       VPM INDEX OF THE FIRST VIRTUAL          */
/*                     PROCESS ASSIGNED TO THIS REAL CPU.      */
/*     VP$END:         INDEX IN VPM OF LAST VIRTUAL...         */
/*     VPS$PER$CPU:    THE NUMBER OF VP ASSIGNED TO THIS       */
/*                     REAL CPU.  MAX IS 10.                   */
/*     LAST$RUN:       VPM INDEX OF THE PROCESS MOST           */
/*                     RECENTLY SWITCHED FROM RUNNING TO       */
/*                     EITHER READY OR WAITING.                */
/*     COUNTER:        AN ARBITRARY MEASURE OF PERFORMANCE.    */
/*                     COUNT MADE WHILE IN IDLE STATE.         */

DECLARE PRDS STRUCTURE
   (CPU$NUMBER         BYTE,
    VP$START           BYTE,
    VP$END             BYTE,
    VPS$PER$CPU        BYTE,
    LAST$RUN           BYTE,
    COUNTER            WORD) PUBLIC INITIAL(0,0,0,0,0,0);

/*0112*********************************************************/
/* GLOBAL DATA BASE DECLARATIONS                               */
/*     DECLARED PUBLIC IN FILE   'GLOBAL.SRC'                  */
/*                      IN MODULE 'GLOBAL$MODULE'              */

DECLARE VPM( MAX$CPU$$$$MAX$VPS$CPU ) STRUCTURE
        (VP$ID          BYTE,
         STATE          BYTE,
         VP$PRIORITY    BYTE,
         EVC$THREAD     BYTE,
         EVC$AW$VALUE   WORD,
         SP$REG         WORD,
         SS$REG         WORD) EXTERNAL;


      DECLARE
         CPU$INIT       BYTE EXTERNAL,
         HDW$INT$FLAG( MAX$CPU ) BYTE EXTERNAL,
         NR$VPS( MAX$CPU ) BYTE EXTERNAL,
         NR$RPS         BYTE EXTERNAL,
         GLOBAL$LOCK    BYTE EXTERNAL;
```

```
     DECLARE
        EVENTS BYTE EXTERNAL,
        EVC$TBL(100) STRUCTURE
           (EVC$NAME       BYTE,
            VALUE          WORD,
            REMOTE$ADDR    WORD,
            THREAD         BYTE) EXTERNAL;

     DECLARE
        SEQUENCERS BYTE EXTERNAL,
        SEQ$TABLE(100) STRUCTURE
            (SEQ$NAME      BYTE,
             SEQ$VALUE     WORD) EXTERNAL;

/*0148************************************************************/
/* DECLARATION OF EXTERNAL PROCEDURE REFERENCES                  */
/*    THE FILE AND MODULE WHERE THEY ARE DEFINED ARE             */
/*    LISTED.                                                    */


     INITIAL$PROC: PROCEDURE EXTERNAL;  END;
        /* IN FILE:    INITKK.SRC */
        /* IN MODULE:  INIT$MOD   */

     AWAIT: PROCEDURE (EVC$ID,AWAITED$VALUE) EXTERNAL;
        DECLARE EVC$ID BYTE, AWAITED$VALUE WORD;
     END;

     VPSCHEDULER: PROCEDURE EXTERNAL;  END;
        /* IN FILE:    SCHED.ASM */

     DECLARE INTVEC LABEL EXTERNAL;
        /* IN FILE:    SCHED.ASM */

     DECLARE INTR$VECTOR POINTER AT(0110H) INITIAL(@INTVEC);
        /* IN FILE:    SCHED.ASM */

/*0171************************************************************/
/*  THESE DIAGNOSTIC MESSAGES MAY EVENTUALLY BE REMOVED.         */
/*  THE UTILITY PROCEDURES, HOWEVER, ARE ALSO USED BY THE        */
/*  MONITOR PROCESS.  THEY SHOULD NOT BE REMOVED.                */


$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
```

```
DECLARE

   MSG1(*)  BYTE INITIAL ('ENTERING RET$VP  ',13,10,'%'),
   MSG1A(*) BYTE INITIAL ('   RUNNING$VP$INDEX =  %'),
   MSG4(*)  BYTE INITIAL ('ENTERING RDYTHISVP',13,10,'%'),
   MSG4A(*) BYTE INITIAL ('   SET VP TO READY:    VP =  %'),
   MSG7(*)  BYTE INITIAL ('ENTERING GETWORK',13.10,'%'),
   MSG7A(*) BYTE INITIAL ('   SET VP TO RUNNING:  VP =  %'),
   MSG7B(*) BYTE INITIAL ('   SELECTED$DBR = %'),
   MSG10(*) BYTE INITIAL ('ENTERING IDLE$VP ',13,10,'%'),
   MSG11(*) BYTE INITIAL ('UPDATE IDLE COUNT ',13,10,'%'),
   MSG12(*) BYTE INITIAL ('ENTERING KERNEL$INIT',10,13,'%'),
   MSG20(*) BYTE INITIAL ('ENTERING LOCATE$EVC ',10,13,'%'),
   MSG22(*) BYTE INITIAL ('ENTERING LOCATE$SEQ ',10,13,'%'),
   MSG23(*) BYTE INITIAL ('   FOUND',10,13,'%'),
   MSG24(*) BYTE INITIAL ('   NOT FOUND',10,13,'%');

DECLARE
   CR LITERALLY '0DH',
   LF LITERALLY '0AH';

OUT$CHAR: PROCEDURE( CHAR ) EXTERNAL;
   DECLARE CHAR BYTE;
END;

OUT$LINE: PROCEDURE( LINE$PTR ) EXTERNAL;
   DECLARE LINE$PTR POINTER;
END;

OUT$NUM: PROCEDURE( NUM ) EXTERNAL;
   DECLARE NUM BYTE;
END;

OUT$DNUM: PROCEDURE( DNUM ) EXTERNAL;
   DECLARE DNUM WORD;
END;

OUT$HEX: PROCEDURE(B) EXTERNAL;
   DECLARE B BYTE;
END;

IN$CHAR: PROCEDURE ( RET$PTR )  EXTERNAL;
   DECLARE RET$PTR POINTER;
END;

IN$DNUM:  PROCEDURE (RET$PTR) EXTERNAL;
   DECLARE RET$PTR POINTER;
END;
IN$NUM: PROCEDURE (RET$PTR) EXTERNAL;
   DECLARE RET$PTR POINTER;
END;
```

```
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

$ENDIF



/*0245*************************************************************/
/* STACK DATA & INITIALIAZTION FOR SYSTEM PROCESSES              */

      DECLARE IDLE$STACK  STRUCTURE
           (LENGTH(030H)     WORD,
           RET$TYPE          WORD,
           BP                WORD,
           DI                WORD,
           SI                WORD,
           DS                WORD,
           DX                WORD,
           CX                WORD,
           AX                WORD,
           BX                WORD,
           ES                WORD,
           START            POINTER,  /* IP,CS */
           FL                WORD) AT(IDLE$STACK$ABS)
                 INITIAL(
       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
       INT$RETURN,7AH,0,0,0,0,0,0,0,0,@IDLE$PROC,200H );

       DECLARE INIT$STACK  STRUCTURE
           (LENGTH(030H)     WORD,
           RET$TYPE          WORD,
           BP                WORD,
           DI                WORD,
           SI                WORD,
           DS                WORD,
           DX                WORD,
           CX                WORD,
           AX                WORD,
           BX                WORD,
           ES                WORD,
           START            POINTER,   /* IP,CS */
           FL                WORD) AT(INIT$STACK$ABS)
              INITIAL(
       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
       INT$RETURN,7AH,0,0,0,0,0,0,0,0,@INITIAL$PROC,200H );
                                /* 200H SETS THE IF FLAG */

$IF NOT MCORTEX
```

```
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

          DECLARE MONITOR$STACK STRUCTURE
              (LENGTH(030H)       WORD,
              RET$TYPE            WORD,
              BP                  WORD,
              DI                  WORD,
              SI                  WORD,
              DS                  WORD,
              DX                  WORD,
              CX                  WORD,
              AX                  WORD,
              BX                  WORD,
              ES                  WORD,
              START               POINTER,
              FL                  WORD) AT(MONITOR$STACK$ABS)
                 INITIAL(
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
       INT$RETURN,7AH,0,0,0,0,0,0,0,0,@MONITOR$PROC,200H );

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

$ENDIF


/*0318*****************************************************/
/**********************************************************/
/*  RET$VP   PROCEDURE                     BREWER 8-18-84  */
/*--------------------------------------------------------*/
/* USED BY THE SCHEDULER TO FIND OUT WHAT IS THE CURRENT   */
/* RUNNING PROCESS.  IT'S INDEX IN VPM IS RETURNED.        */
/*--------------------------------------------------------*/
/* CALLS MADE TO:  OUT$HEX      OUT$CHAR                   */
/**********************************************************/

      RET$VP: PROCEDURE BYTE REENTRANT PUBLIC;

         DECLARE RUNNING$VP$INDEX BYTE;

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

         CALL OUT$LINE(@MSG1);

$ENDIF
```

```
        /*  SEARCH THE VP MAP FOR RUNNING PROCESS INDEX */
        DO   RUNNING$VP$INDEX = PRDS.VP$START TO PRDS.VP$END;
           IF VPM( RUNNING$VP$INDEX ).STATE = RUNNING
           THEN GO TO FOUND;
        END; /* DO */
        RUNNING$VP$INDEX = PRDS.LAST$RUN;

     FOUND:

     $IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

        CALL OUT$LINE(@MSG1A);
        CALL OUT$HEX(RUNNING$VP$INDEX);
        CALL OUT$CHAR(CR);
        CALL OUT$CHAR(LF);

$ENDIF
        RETURN RUNNING$VP$INDEX;
     END; /* RET$VP PROCEDURE */



/*0366***************************************************/
/*  RDYTHISVP      PROCEDURE                BREWER 9-18-84   */
/*-------------------------------------------------------*/
/*  CHANGES A VIRTUAL PROCESSOR STATE TO READY            */
/*-------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$HEX    OUT$CHAR                   */
/*********************************************************/

     RDYTHISVP: PROCEDURE REENTRANT PUBLIC;

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

        CALL OUT$LINE(@MSG4);

$ENDIF

        PRDS.LAST$RUN = RET$VP; /* SAVE INDEX */

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

        CALL OUT$LINE(@MSG4A);
```

```
         CALL OUT$HEX(PRDS.LAST$RUN);
         CALL OUT$CHAR(CR);
         CALL OUT$CHAR(LF);

$ENDIF

         VPM(PRDS.LAST$RUN).STATE = READY;
         RETURN;
      END; /* RDYTHISVP PROCEDURE */


/*0404**************************************************************/
/*  SAVECONTEXT PROCEDURE                      BREWER 8-18-84    */
/*--------------------------------------------------------------*/
/*  SAVES CURRENT STACK POINTER AND SEGMENT IN VPM               */
/*--------------------------------------------------------------*/
/*  CALLS MADE TO: RET$VP                                         */
/****************************************************************/

SAVECONTEXT: PROCEDURE (STACK$PTR, STACK$SEG) REENTRANT
             PUBLIC;

   DECLARE (STACK$PTR, STACK$SEG) WORD;

   IF PRDS.LAST$RUN <> 255 THEN DO; /* IF ENTRY IS NOT */
                                    /* FROM KORE START */
      VPM(PRDS.LAST$RUN).SP$REG = STACK$PTR; /*SAVE STACK*/
      VPM(PRDS.LAST$RUN).SS$REG = STACK$SEG; /* STATE    */
   END;

      END;


/*0426**************************************************************/
/*  GET$SP      PROCEDURE                     BREWER 8-18-84 */
/*--------------------------------------------------------------*/
/*  RETURNS STACK POINTER OF CURRENT RUNNING PROCESS AS          */
/*  SAVED IN THE VIRTUAL PROCESSOR MAP                            */
/*--------------------------------------------------------------*/
/*  CALLS MADE TO: RET$VP                                         */
/****************************************************************/

GET$SP: PROCEDURE WORD REENTRANT PUBLIC;

   DECLARE N BYTE;

   N = RET$VP; /* GET CURRENT RUNNING VIRTUAL PROCESSOR */

   RETURN VPM(N).SP$REG; /* RETURN NEW VP STACK POINTER  */

END;
```

```
/*0415***********************************************************/
/*  GETWORK    PROCEDURE                          BREWER 8-18-84 */
/*---------------------------------------------------------------*/
/*  DETERMINES THE NEXT ELIGIBLE VIRTUAL PROCESSOR TO RUN */
/*---------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$CHAR   OUT$LINE   OUT$DNUM           */
/*****************************************************************/

GETWORK: PROCEDURE WORD REENTRANT PUBLIC;

         DECLARE (PRI,N,I)     BYTE;
         DECLARE SELECTED$DBR WORD;
         DECLARE DISPLAY       BYTE;

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

         CALL OUT$LINE(@MSG7);

$ENDIF


         PRI = 255;
         DO /* SEARCH VPM FOR ELIGIBLE VIRTUAL PROCESSOR
              TO RUN */
            I = PRDS.VP$START TO PRDS.VP$END;
            IF /* THIS VP'S PRIORITY IS HIGHER THAN PRI */
               ((VPM(I).VP$PRIORITY <= PRI) AND
               (VPM(I).STATE = READY))  THEN  DO;
                  /* SELECT THIS VIRTUAL PROCESSOR */
                  PRI = VPM(I).VP$PRIORITY;
                  N = I;
            END; /* IF */
         END; /* DO LOOP SEARCH OF VPM */

         /*  SET SELECTED VIRTUAL PROCESSOR  */
         VPM(N).STATE = RUNNING;
         SELECTED$DBR = VPM(N).SS$REG;

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
         CALL OUT$LINE(@MSG7A);
         CALL OUT$HEX(N);
         CALL OUT$CHAR(CR);
         CALL OUT$CHAR(LF);
         CALL OUT$LINE(@MSG7B);
         CALL OUT$DNUM(SELECTED$DBR);
         CALL OUT$CHAR(CR);
         CALL OUT$CHAR(LF);

$ENDIF
```

```
        RETURN SELECTED$DBR;

END; /* GETWORK PROCEDURE */


/*0507***************************************************************/
/*  LOCATE$EVC PROCEDURE                              BREWER 8-18-84 */
/*-------------------------------------------------------------------*/
/* FUNCTION CALL.  RETURNS THE INDEX IN EVENTCOUNT TABLE            */
/* OF THE EVENT NAME PASSED TO IT.                                   */
/*-------------------------------------------------------------------*/
/* CALLS MADE TO:  OUT$CHAR   OUT$LINE                               */
/*********************************************************************/

LOCATE$EVC: PROCEDURE(EVENT$NAME) BYTE REENTRANT PUBLIC;

   DECLARE EVENT$NAME BYTE;
   DECLARE (MATCH,EVCTBL$INDEX) BYTE;

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
        CALL OUT$LINE(@MSG20);

$ENDIF

   MATCH = FALSE;
   EVCTBL$INDEX = 0;
   /* SEARCH DOWN THE EVENTCOUNT TABLE TO LOCATE THE  */
   /* DESIRED EVENTCOUNT BY MATCHING THE NAMES */
   DO WHILE (MATCH = FALSE)  AND  (EVCTBL$INDEX < EVENTS);
   /* DO WHILE HAVE NOT FOUND THE EVENTCOUNT AND HAVE NOT */
   /* REACHED END OF THE TABLE */
      IF EVENT$NAME = EVC$TBL(EVCTBL$INDEX).EVC$NAME THEN
         MATCH = TRUE;
      ELSE
         EVCTBL$INDEX = EVCTBL$INDEX+1;
   END; /* WHILE */
   /* IF HAVE FOUND THE EVENTCOUNT */
   IF (MATCH = TRUE) THEN DO;
      /* RETURN ITS INDEX IN THE EVC$TBL */

$IF NOT MCORTEX
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
      CALL OUT$LINE(@MSG23);

$ENDIF

      RETURN EVCTBL$INDEX;
   END;
   ELSE DO;
```

```
          /* RETURN NOT FOUND CODE */

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
        CALL OUT$LINE(@MSG24);

$ENDIF

        RETURN NOT$FOUND;
     END; /* ELSE */
END; /* LOCATE$FVC PROCEDURE */


/*0570*****************************************************/
/*  LOCATE$SEQ PROCEDURE                    BREWER 8-18-84 */
/*---- --------- -------------------------------- ---------*/
/* FUNCTION CALL TO RETURN THE INDEX OF THE SEQUENCER      */
/* SPECIFIED IN THE SEQ-TABLE.                             */
/*---------------------------------------------------------*/
/* CALLS MADE TO:  OUT$LINE                                */
/**********************************************************/

LOCATE$SEQ:  PROCEDURE(SEQ$NAME) BYTE REENTRANT PUBLIC;

   DECLARE SEQ$NAME BYTE;
   DECLARE ( MATCH, SEQTBL$INDEX ) BYTE;

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

   CALL OUT$LINE(@MSG22);

$ENDIF


   MATCH = FALSE;
   SEQTBL$INDEX = 0;
   DO WHILE (MATCH = FALSE) AND (SEQTBL$INDEX < SEQUENCERS);
      IF SEQ$NAME = SEQ$TABLE(SEQTBL$INDEX).SEQ$NAME THEN
         MATCH = TRUE;
      ELSE
         SEQTBL$INDEX = SEQTBL$INDEX + 1;
   END; /* WHILE */
   IF (MATCH = TRUE) THEN DO;

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
      CALL OUT$LINE(@MSG23);
```

```
$ENDIF

      RETURN SEQTBL$INDEX;
   END; /* IF */
   ELSE DO;

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
      CALL OUT$LINE(@MSG24);

$ENDIF

      RETURN NOT$FOUND;
   END; /* ELSE */
END; /* LOCATE$SEQ PROCEDURE */
```

```
/*0629*************************************************************/
/******************************************************************/
/*  SYSTEM PROCESSES                                              */
/*                                                                */


/******************************************************************/
/*  IDLE PROCESS                                 BREWER 8-18-84   */
/*----------------------------------------------------------------*/
/*  THIS PROCESS IS SCHEDULED IF ALL OTHER PROCESSES IN           */
/*  THE VPM ARE BLOCKED.  THE STARTING ADDRESS IS PROVIDED*/
/*  TO THE IDLE$STACK AND PLACED IN PRDS.IDLE$DBR.   A            */
/*  CALL TO THE SCHEDULER IS MADE EVERY 4 MS IN THE               */
/*  EVENT THAT AN ONBOARD PROCESS WAS READIED BY AN               */
/*  OFFBOARD OPERATION (ADVANCE). EVERY 250 ITERATIONS            */
/*  THE COUNT IS INCREMENTED BY ONE. THUS, THE COUNT IS           */
/*  INCREMENTED ONCE PER SECOND. THE COUNT IS MAINTAINED          */
/*  IN THE PRDS TABLE AND IS A ROUGH MEASURE OF SYSTEM            */
/*  PERFORMANCE BY GIVING AN INDICATION OF THE AMOUNT OF          */
/*  TIME SPENT IN THE IDLE PROCESS.                               */
/*----------------------------------------------------------------*/
/*  CALLS MADE TO:  PLM86 PROCEDURE 'TIME'                        */
/*                  OUT$LINE                                      */
/******************************************************************/

IDLE$PROC:  PROCEDURE REENTRANT PUBLIC;

         DECLARE I BYTE;

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
         CALL OUT$LINE(@MSG10);

$ENDIF


       LOOP: DO I = 1 TO 250;
        /* 4 MS DELAY */
               CALL TIME( 10 );
               DO WHILE LOCK$SET(@GLOBAL$LOCK, 119);
               /* ASSERT LOCK */
               END;
               CALL RDYTHISVP;
               CALL VPSCHEDULER;
             END;   /* DO I */

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
            CALL OUT$LINE(@MSG11);

$ENDIF
```

```
          PRDS.COUNTER = PRDS.COUNTER + 1;
          GO TO LOOP;
      END;   /* IDLE$PROC  */


/*0691*************************************************************/
/*  MONITOR PROCESS                               BREWER 8-18-84  */
/*----------------------------------------------------------------*/
/*  THE MONITOR PROCESS IS INITIALIZED BY THE OS LIKE             */
/*  INIT AND IDLE.  IT HAS THE RESERVED ID OF 0FEH AND A          */
/*  PRIORITY OF 0H.  IT IS ALWAYS BLOCKED OR WAITING UNTIL*/
/*  IT IS PREEMTED BY THE USER.                                   */
/*----------------------------------------------------------------*/
/*  CALLS MADE TO:  OUT$LINE        OUT$CHAR                      */
/*                  OUT$DNUM        IN$DNUM                       */
/*                  IN$NUM                                        */
/******************************************************************/

$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
MONITOR$PROC:  PROCEDURE REENTRANT PUBLIC;

   DECLARE
      PTR                  POINTER,
      PTR2                 POINTER,
      PTR3 BASED PTR2      POINTER,
      ADDR STRUCTURE (OFFSET WORD, BASE WORD),
      CONTENTS BASED PTR BYTE;

   DECLARE
      (LINECOMPLETE, LOOP2)      BYTE,
      (QUANTITY, COUNT)          BYTE,
      (INCHR, INDEX, VALID$CMD)  BYTE;

   LOOP:  VALID$CMD = 0;

   CALL OUT$CHAR(CR);
   CALL OUT$CHAR(LF);
   CALL OUT$CHAR('.');
   DO WHILE NOT VALID$CMD;
      CALL IN$CHAR(@INCHR);
      IF (INCHR = 'D') OR (INCHR = 'S') OR (INCHR = 'E') THEN
         VALID$CMD = 0FFH;
         IF (INCHR=64H) OR (INCHR=65H) OR (INCHR=73H) THEN
           VALID$CMD = 0FFH;
         IF VALID$CMD = 0FFH THEN CALL OUT$CHAR(INCHR);
   END; /* DO WHILE */
   IF (INCHR = 'D') OR (INCHR = 64H) THEN DO;
      /* DISPLAY COMMAND SECTION */
      CALL IN$DNUM(@ADDR.BASE);
      CALL OUT$CHAR(':');
      CALL IN$DNUM(@ADDR.OFFSET);
      PTR2 = @ADDR;
      PTR = PTR3;
      /* CONTENTS SHOULD NOW BE SET */
      DO WHILE (INCHR<>CR) AND (INCHR<>23H);
         CALL IN$CHAR(@INCHR);
      END; /* DO WHILE */
      IF INCHR = CR THEN DO;
         CALL OUT$CHAR('-');
         CALL OUT$NUM(CONTENTS);
         CALL OUT$CHAR(CR);
         CALL OUT$CHAR(LF);
      END;     /* IF NORMAL 1 ADDR DISPLAY */
      IF INCHR = 23H THEN DO;
         COUNT = 0;
         CALL OUT$CHAR('#');
         CALL IN$NUM(@QUANTITY);
         DO WHILE QUANTITY > 0;
           CALL OUT$CHAR(CR);
           CALL OUT$CHAR(LF);
           CALL OUT$DNUM(ADDR.BASE);
           CALL OUT$CHAR(':');
           CALL OUT$DNUM(ADDR.OFFSET);
           LINECOMPLETE = FALSE;
           DO WHILE LINECOMPLETE = FALSE;
              CALL OUT$CHAR(' ');
              CALL OUT$NUM(CONTENTS);
              ADDR.OFFSET = ADDR.OFFSET + 1;
              PTR = PTR3;
              QUANTITY = QUANTITY - 1;
              IF ((ADDR.OFFSET AND 000FH)=0) OR
                  (QUANTITY = 0) THEN LINECOMPLETE=TRUE;
           END; /* DO WHILE LINE NOT COMPLETE */
         END; /* DO WHILE QUANTITY */
      END; /* IF MULTI ADDR DISPLAY */
   END; /* DISPLAY COMMAND SECTION */
   IF (INCHR='S') OR (INCHR=73H) THEN DO;
        /* SUBSTITUTE COMMAND SECTION */
      CALL IN$DNUM(@ADDR.BASE);
      CALL OUT$CHAR(':');
      CALL IN$DNUM(@ADDR.OFFSET);
      CALL OUT$CHAR('-');

      PTR2 = @ADDR;
      PTR = PTR3;
         /* CURRENT CONTENTS SHOULD NOW BE AVAILABLE */
      CALL OUT$NUM(CONTENTS);
      LOOP2 = TRUE;
      DO WHILE LOOP2 = TRUE;
         DO WHILE (INCHR<>',')AND(INCHR<>' ')
                   AND(INCHR<>CR);
              CALL IN$CHAR(@INCHR);
         END;
         IF (INCHR = CR) THEN LOOP2 = FALSE;
         IF (INCHR = ',') THEN DO;
```

```
            /* SKIP THIS ADDR AND GO TO NEXT FOR SUB */
          CALL OUT$CHAR(CR);
          CALL OUT$CHAR(LF);
          ADDR.OFFSET = ADDR.OFFSET + 1;
          PTR = PTR3;
          CALL OUT$DNUM(ADDR.BASE);
          CALL OUT$CHAR(':');
          CALL OUT$DNUM(ADDR.OFFSET);
          CALL OUT$CHAR('-');
          CALL OUT$NUM(CONTENTS);
        END;  /* IF SKIP FOR NEXT SUB */
        IF (INCHR = ' ') THEN DO;
          CALL OUT$CHAR(' ');
          CALL IN$NUM(@CONTENTS);
          DO WHILE (INCHR<>CR)AND(INCHR<>',');
             CALL IN$CHAR(@INCHR);
          END;
          IF (INCHR = CR) THEN LOOP2 = FALSE;
          IF (INCHR = ',') THEN DO;
             CALL OUT$CHAR(',');
             ADDR.OFFSET = ADDR.OFFSET + 1;
             PTR = PTR3;
             CALL OUT$CHAR(CR);
             CALL OUT$CHAR(LF);
             CALL OUT$DNUM(ADDR.BASE);
             CALL OUT$CHAR(':');
             CALL OUT$DNUM(ADDR.OFFSET);
             CALL OUT$CHAR('-');
             CALL OUT$NUM(CONTENTS);
          END; /* IF GO TO NEXT ADDR */
        END; /* IF CHANGE CONTENTS */
        INCHR = 'X'; /* REINITIALIZE CMD */
      END; /* LOOP, CONTINUOUS SUB CMD */
   END; /* SUBSTITUTE COMMAND SECTION */

   IF (INCHR='E') OR (INCHR=65H) THEN DO;
        /* FIND OUT WHICH VPS IS RUNNING 'ME' */
      INDEX = RET$VP;
      /* NOW BLOCK MYSELF */
      DISABLE;
      PRDS.LAST$RUN = INDEX;
      VPM(INDEX).STATE = WAITING;
      CALL VPSCHEDULER; /* NO RETURN */
   END; /* IF */
   GO TO LOOP;
END; /* MONITOR PROCESS */
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

$ENDIF


/*0347***********************************************************/
/****************************************************************/
```

```
/*          STARTING POINT OF THE OPERATING SYSTEM              */
/*--------------------------------------------------------------*/
/* ROUTINE INITIALIZES THE OS AND IS NOT REPEATED.              */
/****************************************************************/
/****************************************************************/
/* TO INITIALIZE THE PRDS TABLE FOR THIS CPU */
DECLARE CPU$PTR POINTER DATA(@PRDS.CPU$NUMBER),
        ZZ BYTE;

   DISABLE;


$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
   CALL OUT$LINE(@MSG12);

$ENDIF


/* INITIALIZE   P P I   AND   P I C */
   OUTPUT(PORT$CE) = 0C0H;/* PPI - MICROPOLIS + MCORTEX */
   OUTPUT(PORT$C0) = 13H;  /* PIC - ICW1 - EDGE TRIGGERED */
   OUTPUT(PORT$C2) = 40H;/* PIC-ICW2-VECTOR TABLE ADDRESS */
   OUTPUT(PORT$C2) = 0FH;/* PIC-ICW4-MCS86 MODE, AUTO EOI */
   OUTPUT(PORT$C2) = 0AFH; /*PIC-MASK ALLOWING INT. 4 & 6 */



/* ESTABLISH UNIQUE SEQUENTIAL NUMBER FOR THIS CPU */
/* SET GLOBAL$LOCK */
   DO WHILE LOCK$SET(@GLOBAL$LOCK,119);  END;
   PRDS.CPU$NUMBER = CPU$INIT;
   CPU$INIT = CPU$INIT + 1;

/* RELEASE GLOBAL LOCK */
   GLOBAL$LOCK = 0;

/*  SET UP INITIAL START AND END FOR PROC TABLE */
   PRDS.VP$START = 0;
   DO ZZ = 1 TO PRDS.CPU$NUMBER;
     PRDS.VP$START = PRDS.VP$START + MAX$VPS$CPU;
   END;

$IF MCORTEX

/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
   PRDS.VP$END = PRDS.VP$START + 1;
   PRDS.VPS$PER$CPU = 2;
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/

$ELSE
```

```
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
   PRDS.VP$END = PRDS.VP$START + 2;
   PRDS.VPS$PER$CPU = 3;
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

$ENDIF


   /* INITIALIZE THE VP MAP FOR IDLE AND INIT PROC */
   /* AND MONITOR PROCESS */
   VPM(PRDS.VP$START).VP$ID = 255;
   VPM(PRDS.VP$START).STATE = 1;
   VPM(PRDS.VP$START).VP$PRIORITY = 0;
   VPM(PRDS.VP$START).EVC$THREAD = 255;
   VPM(PRDS.VP$START).EVC$AW$VALUE = 0;
   VPM(PRDS.VP$START).SP$REG = 60H;
   VPM(PRDS.VP$START).SS$REG = INIT$STACK$SEG;
   VPM(PRDS.VP$START+1).VP$ID = 255;
   VPM(PRDS.VP$START+1).STATE = 1;
   VPM(PRDS.VP$START+1).VP$PRIORITY = 255;
   VPM(PRDS.VP$START+1).EVC$THREAD = 255;
   VPM(PRDS.VP$START+1).EVC$AW$VALUE = 0;
   VPM(PRDS.VP$START+1).SP$REG = 60H;
   VPM(PRDS.VP$START+1).SS$REG = IDLE$STACK$SEG;


$IF NOT MCORTEX

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
   VPM(PRDS.VP$START+2).VP$ID = 0FEH;
   VPM(PRDS.VP$START+2).STATE = 7;
   VPM(PRDS.VP$START+2).VP$PRIORITY = 0;
   VPM(PRDS.VP$START+2).EVC$THREAD = 255;
   VPM(PRDS.VP$START+2).EVC$AW$VALUE = 0;
   VPM(PRDS.VP$START+2).SP$REG = 60H;
   VPM(PRDS.VP$START+2).SS$REG = MONITOR$STACK$SEG;

/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/


$ENDIF

   NR$RPS = NR$RPS + 1;

$IF MCORTEX

/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/
   NR$VPS(PRDS.CPU$NUMBER) = 2;
/**** MCORTEX **** MCORTEX ***** MCORTEX **** MCORTEX ****/

$ELSE
```

```
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
   NR$VPS(PRDS.CPU$NUMBER) = 3;
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/
/**** MXTRACE **** MXTRACE ***** MXTRACE **** MXTRACE ****/

$ENDIF

   HDW$INT$FLAG( PRDS.CPU$NUMBER ) = 0 ;
   ENABLE;

   PRDS.LAST$RUN = 255; /* INDICATE START ENTRY TO
                           SCHEDULER */
   CALL VPSCHEDULER;         /* - - NO RETURN */

/**************************************************************/
/**************************************************************/

END;  /* L1$MODULE */

/**************************************************************/
/**************************************************************/
/**************************************************************/
*****     MCORTEX        MCORTEX       MCORTEX             *****

ISIS-II MCS-86 LINKER, V1.1, INVOKED BY:
:F1:LINK86 :F1:LEVEL1.OBJ,:F1:LEVEL2.OBJ,:F1:SCHED.OBJ,&
:F1:INITK.OBJ,:F1:GLOBAL.OBJ TO :F1:KORE.LNK
LINK MAP FOR :F1:KORE.LNK(L1MODULE)

LOGICAL SEGMENTS INCLUDED:
LENGTH ADDRESS  SEGMENT          CLASS
 03D3H  ------  L1MODULE_CODE    CODE
 0008H  ------  L1MODULE_DATA    DATA
 004CH  ------  STACK            STACK
 0000H  ------  MEMORY           MEMORY
 09C4H  ------  L2MODULE_CODE    CODE
 000CH  -----   L2MODULE_DATA    DATA
 0000H  ------  ??SEG
 0097H  ------  SCHEDULER
 001AH  ------  INITMOD_CODE     CODE
 0001H  ------  INITMOD_DATA     DATA
 0000H  ------  GLOBALMODULE_C   CODE
               -ODE
 0737H  ------  GLOBALMODULE_D   DATA
               -ATA

INPUT MODULES INCLUDED:
:F1:LEVEL1.OBJ(L1MODULE)
:F1:LEVEL2.OBJ(L2MODULE)
:F1:SCHED.OBJ(SCHED)
:F1:INITK.OBJ(INITMOD)
:F1:GLOBAL.OBJ(GLOBALMODULE)
```

```
ISIS-II MCS-86 LOCATER, V1.1 INVOKED BY:
:F1:LOC86 :F1:KORE.LNK ADDRESSES(SEGMENTS(&
STACK(ØC55ØH),&
INITMOD_CODE(0439ØH),&
GLOBALMODULE_DATA(ØE53ØØH)))&
SEGSIZE(STACK(75H))&
RESERVE(ØH TO ØB6FFH)
WARNING 56:  SEGMENT IN RESERVED SPACE
  SEGMENT:  (NO NAME)
WARNING 56:  SEGMENT IN RESERVED SPACE
  SEGMENT:  INITMOD_CODE

SYMBOL TABLE OF MODULE L1MODULE
READ FROM FILE :F1:KORE.LNK
WRITTEN TO FILE :F1:KORE

BASE   OFFSET TYPE SYMBOL          BASE   OFFSET TYPE SYMBOL

ØC49H  ØØØ8H  PUB  PRDS            ØB7ØH  Ø38ØH  PUB  IDLEPROC
ØB7ØH  Ø3Ø2H  PUB  LOCATESEQ       ØB7ØH  Ø284H  PUB  LOCATEEVC
ØB7ØH  Ø2ØBH  PUB  GETWORK         ØB7ØH  Ø1E3H  PUB  GETSP
ØB7ØH  Ø1AEH  PUB  SAVECONTEXT     ØB7ØH  Ø185H  PUB  RDYTHISVP
ØB7ØH  Ø13AH  PUB  RETVP           ØBADH  Ø977H  PUB  DISTRI-
                                                    BUTIONMAP
ØBADH  Ø953H  PUB  DEFINECLUSTER   ØBADH  Ø814H  PUB  SYSTEMIO
ØBADH  Ø6AFH  PUB  CREATEPROC      ØBADH  Ø64EH  PUB  TICKET
ØBADH  Ø5EBH  PUB  CREATESEQ       ØBADH  Ø3F3H  PUB  PREEMPT
ØBADH  Ø25AH  PUB  ADVANCE         ØBADH  Ø1AAH  PUB  AWAIT
ØBADH  Ø159H  PUB  READ            ØBADH  ØØE3H  PUB  CREATEEVC
ØBADH  ØØ36H  PUB  GATEKEEPER      ØC4BH  ØØØØH  PUB  VPSCHEDULER
ØC4BH  ØØ33H  PUB  INTVEC          Ø439H  ØØØ2H  PUB  INITIALPROC
E53ØH  Ø25AH  PUB  VPM             E53ØH  Ø65BH  PUB  SEQTABLE
E53ØH  Ø65AH  PUB  SEQUENCERS      E53ØH  Ø659H  PUB  CPUINIT
E53ØH  ØØØ2H  PUB  EVCTBL          E53ØH  ØØØØH  PUB  LOCAL-
                                                    CLUSTERADDR
E53ØH  Ø658H  PUB  EVENTS          E53ØH  Ø64EH  PUB  HDWINTFLAG
E53ØH  Ø644H  PUB  NRVPS           E53ØH  Ø643H  PUB  NRRPS
E53ØH  Ø642H  PUB  GLOBALLOCK
```

```
MEMORY MAP OF MODULE L1MODULE
READ FROM FILE :F1:KORE.LNK
WRITTEN TO FILE :F1:KORE

MODULE START ADDRESS  PARAGRAPH = 0B70H  OFFSET = 0030H
SEGMENT MAP

START     STOP      LENGTH ALIGN NAME              CLASS

00110H    00113H     0004H   A   (ABSOLUTE)
04390H    043A9H     001AH   W   INITMOD_CODE      CODE
0B700H    0BAD2H     03D3H   W   L1MODULE_CODE     CODE
0BAD4H    0C497H     09C4H   W   L2MODULE_CODE     CODE
0C498H    0C498H     0000H   W   GLOBALMODULE_C    CODE
                                -ODE
0C498H    0C49FH     0008H   W   L1MODULE_DATA     DATA
0C4A0H    0C4A0H     0000H   W   L2MODULE_DATA     DATA
0C4A0H    0C4A0H     0001H   W   INITMOD_DATA      DATA
0C4B0H    0C4B0H     0000H   G   ??SEG
0C4B0H    0C546H     0097H   G   SCHEDULER
0C550H    0C5C4H     0075H   W   STACK             STACK
0C5D0H    0C649H     007AH   A   (ABSOLUTE)
0C650H    0C6C9H     007AH   A   (ABSOLUTE)
10000H    10077H     0078H   A   (ABSOLUTE)
E5300H    E5A86H     0787H   W   GLOBALMODULE_D    DATA
                                -ATA
E5A88H    E5A88H     0000H   W   MEMORY            MEMORY
```

```
ISIS-II MCS-86 LINKER, V1.1, INVOKED BY:
:F1:LINK86 :F1:LEVEL1.OBJ,:F1:LEVEL2.OBJ,:F1:SCHED.OBJ,&
:F1:INITK.OBJ,:F1:GLOBAL.OBJ TO :F1:KORE.LNK
LINK MAP FOR :F1:KORE.LNK(L1MODULE)

LOGICAL SEGMENTS INCLUDED:
LENGTH ADDRESS  SEGMENT            CLASS
 08C6H  ------  L1MODULE_CODE      CODE
 0133H  ------  L1MODULE_DATA      DATA
 0062H  ------  STACK              STACK
 0000H  ------  MEMORY             MEMORY
 0DFFH  ------  L2MODULE_CODE      CODE
 00F5H  ------  L2MODULE_DATA      DATA
 0000H  ------  ??SEG
 0097H  ------  SCHEDULER
 001AH  ------  INITMOD_CODE       CODE
 0001H  ------  INITMOD_DATA       DATA
 0000H  ------  GLOBALMODULE_C     CODE
               -ODE
 0787H  ------  GLOBALMODULE_D     DATA
               -ATA

INPUT MODULES INCLUDED:
:F1:LEVEL1.OBJ(L1MODULE)
:F1:LEVEL2.OBJ(L2MODULE)
:F1:SCHED.OBJ(SCHED)
:F1:INITK.OBJ(INITMOD)
:F1:GLOBAL.OBJ(GLOBALMODULE)
```

```
ISIS-II MCS-86 LOCATER, V1.1 INVOKED BY:
:F1:LOC86 :F1:KORE.LNK ADDRESSES(SEGMENTS(&
STACK(ØC5BØH),&
INITMOD_CODE(Ø439ØH),&
GLOBALMODULE_DATA(ØE53ØØH)))&
SEGSIZE(STACK(75H))&
RESERVE(ØH TO ØABFFH)
WARNING 56:  SEGMENT IN RESERVED SPACE
  SEGMENT:  (NO NAME)
WARNING 56:  SEGMENT IN RESERVED SPACE
  SEGMENT:  INITMOD_CODE

SYMBOL TABLE OF MODULE L1MODULE
READ FROM FILE :F1:KORE.LNK
WRITTEN TO FILE :F1:KORE

BASE   OFFSET TYPE SYMBOL          BASE   OFFSET TYPE SYMBOL

ØC2CH  ØØØ6H  PUB  PRDS            ØACØH  Ø5Ø5H  PUB  MONITORPROC
ØACØH  Ø49CH  PUB  IDLEPROC        ØACØH  Ø3FDH  PUB  LOCATESEQ
ØACØH  Ø35EH  PUB  LOCATEEVC       ØACØH  Ø293H  PUB  GETWORK
ØACØH  Ø26BH  PUB  GETSP           ØACØH  Ø236H  PUB  SAVECONTEXT
ØACØH  Ø1DEH  PUB  RDYTHISVP       ØACØH  Ø165H  PUB  RETVP
ØB4CH  ØDD1H  PUB  OUTHEX          ØB4CH  ØCCCH  PUB  INHEX
ØB4CH  ØC7CH  PUB  SENDCHAR        ØB4CH  ØC59H  PUB  RECVCHAR
ØB4CH  ØC2DH  PUB  OUTDNUM         ØB4CH  ØBF4H  PUB  INDNUM
ØB4CH  ØBDCH  PUB  OUTNUM          ØB4CH  ØB8DH  PUB  OUTLINE
ØB4CH  ØB75H  PUB  OUTCHAR         ØB4CH  ØB5AH  PUB  INNUM
ØB4CH  ØB3FH  PUB  INCHAR          ØB4CH  ØAEEH  PUB  DISTRI-
                                                   BUTIONMAP
ØB4CH  ØACAH  PUB  DEFINECLUSTER   ØB4CH  Ø98BH  PUB  SYSTEMIO
ØB4CH  Ø81BH  PUB  CREATEPROC      ØB4CH  Ø7AFH  PUB  TICKET
ØB4CH  Ø729H  PUB  CREATESEQ       ØB4CH  Ø51ØH  PUB  PREEMPT
ØB4CH  Ø36CH  PUB  ADVANCE         ØB4CH  Ø2B1H  PUB  AWAIT
ØB4CH  Ø23DH  PUB  READ            ØB4CH  Ø1A4H  PUB  CREATEEVC
ØB4CH  ØØ68H  PUB  GATEKEEPER      ØC4FH  ØØØØH  PUB  VPSCHEDULER
ØC4FH  ØØ33H  PUB  INTVEC          Ø439H  ØØØ2H  PUB  INITIALPROC
E53ØH  Ø25AH  PUB  VPM             E53ØH  Ø65BH  PUB  SEQTABLE
E53ØH  Ø65AH  PUB  SEQUENCERS      E53ØH  Ø659H  PUB  CPUINIT
E53ØH  ØØØ2H  PUB  EVCTBL          E53ØH  ØØØØH  PUB  LOCAL-
                                                    CLUSTERADDR
E53ØH  Ø658H  PUB  EVENTS          E53ØH  Ø64EH  PUB  HDWINTFLAG
E53ØH  Ø644H  PUB  NRVPS           E53ØH  Ø643H  PUB  NRRPS
E53ØH  Ø642H  PUB  GLOBALLOCK
```

```
MEMORY MAP OF MODULE L1MODULE
READ FROM FILE :F1:KORE.LNK
WRITTEN TO FILE :F1:KORE

MODULE START ADDRESS  PARAGRAPH = ØACØH  OFFSET = ØØ3ØH
SEGMENT MAP

START      STOP       LENGTH ALIGN NAME                CLASS

ØØ11ØH    ØØ113H      ØØØ4H    A    (ABSOLUTE)
Ø439ØH    Ø43A9H      ØØ1AH    W    INITMOD_CODE        CODE
ØACØØH    ØB4C5H      Ø8C6H    W    L1MODULE_CODE       CODE
ØB4C6H    ØC2C4H      ØDFFH    W    L2MODULE_CODE       CODE
ØC2C6H    ØC2C6H      ØØØØH    W    GLOBALMODULE_C      CODE
                                   -ODE
ØC2C6H    ØC3F8H      Ø133H    W    L1MODULE_DATA       DATA
ØC3FAH    ØC4DEH      ØØE5H    W    L2MODULE_DATA       DATA
ØC4EØH    ØC4EØH      ØØØ1H    W    INITMOD_DATA        DATA
ØC4FØH    ØC4FØH      ØØØØH    G    ??SEG
ØC4FØH    ØC586H      ØØ97H    G    SCHEDULER
ØC5BØH    ØC624H      ØØ75H    W    STACK               STACK
ØC63ØH    ØC6A9H      ØØ7AH    A    (ABSOLUTE)
ØC6BØH    ØC729H      ØØ7AH    A    (ABSOLUTE)
ØC73ØH    ØC7A9H      ØØ7AH    A    (ABSOLUTE)
1ØØØØH    1ØØ77H      ØØ78H    A    (ABSOLUTE)
E53ØØH    E5A86H      Ø787H    W    GLOBALMODULE_D      DATA
                                   -ATA
E5A88H    E5A88H      ØØØØH    W    MEMORY              MEMORY
```

# APPENDIX I

## SCHEDULER & INTERRUPT HANDLER SOURCE CODE

The ASM86 code in file SCHED.ASM is part of the LEVEL I module. Details pertaining to assembler invocation may be found in [Ref. 20] and [Ref. 21]. This module is linked into file KORE.LNK and its memory map is included in the map for KORE.

```
;*0007*************************************************************
;*   SCHEDULER      ASM FILE                    BREWER 8-18-84     *
;*-------- --------------------------------------------------------*
;* THE FOLLOWING ARE THE EXTERNAL PLM86 PROCEDURES CALLED         *
;* BY THIS MODULE.                                                 *

   EXTRN SAVECONTEXT:FAR
   EXTRN GETSP:FAR
   EXTRN GETWORK:FAR
   EXTRN RDYTHISVP:FAR
   EXTRN PRDS:BYTE
   EXTRN HDWINTFLAG:BYTE
   EXTRN GLOBALLOCK:BYTE

   SCHEDULER SEGMENT
   PUBLIC VPSCHEDULER
   PUBLIC INTVEC

   VPSCHEDULER PROC FAR

   ASSUME CS:SCHEDULER
   ASSUME DS:NOTHING
   ASSUME SS:NOTHING
   ASSUME ES:NOTHING

   ; ENTRY POINT FOR A CALL TO SCHEDULER

   CLI
   PUSH DS
   MOV CX,0F

   ;SWAP VIRTUAL PROCESSORS.  THIS IS DONE BY SAVING THE
   ;STACK BASE POINTER AND THE RETURN TYPE FLAG ON THE
   ;STACK, AND BY SAVING THE STACK SEGMENT AND STACK
   ;POINTER IN THE VIRTUAL PROCESSOR MAP.

INTJOIN: PUSH BP         ;SAVE "CURRENT" STACK BASE
   PUSH CX                ;SAVE 'CURRENT' IRET_IND FLAG
   MOV AX,SP
   PUSH AX                ;SET UP SAVE$CONTEXT PARAMETERS
   PUSH SS                ;SET UP SAVE$CONTEXT PARAMETERS
   CALL SAVECONTEXT
   CALL GETWORK           ;GET NEW STACK SEGMENT
   PUSH AX                ;TEMPORY SAVE OF STACK SEGMENT
   CALL GETSP             ;GET NEW STACK POINTER
   POP SS                 ;INSTALL NEW STACK SEGMENT
   MOV SP,AX              ;INSTALL NEW STACK POINTER

   ;SWAP VIRTUAL PROCESSOR CONTEXT COMPLETE AT THIS POINT
   ;NOW OPERATING IN NEWLY SELECTED PROCESS STACK
   POP CX                           ;GET IRET_IND FLAG
   POP BP                           ;INSTALL NEW STACK BASE

   ;  CHECK FOR RETURN TYPE, NORMAL OR INTERRUPT

   CMP CX,77H
   JZ  INTRET

NORM_RET: POP DS
    ; UNLOCK GLOBAL$LOCK
   MOV   AX,SEG GLOBALLOCK
   MOV   ES, AX
   MOV   ES:GLOBALLOCK,Ø
   STI
   RET

   VPSCHEDULER ENDP

;******************************************************************


;******************************************************************
;*  INTERRUPT HANDLER                                              *
;*                                                                 *

   INTERRUPT_HANDLER  PROC NFAR

   ASSUME CS:SCHEDULER
   ASSUME DS:NOTHING
   ASSUME SS:NOTHING
   ASSUME ES:NOTHING

INTVEC: CLI
   PUSH ES    ; SAVE NEEDED REGs TO TEST INTERRUPT FLAG
   PUSH BX
   PUSH AX
   PUSH CX
   CALL HARDWARE_INT_FLAG
   MOV   AL,Ø
   XCHG  AL,ES:HDWINTFLAG[BX]
   CMP   AL,77H              ; IS INT FLAG ON ?
   JZ    PUSH_REST_REGS      ; IF 'YES' SAVE REST REGs
   POP   CX                  ; IF 'NOT' RESUME PREVIOUS
   POP   AX                  ; EXECUTION POINT
   POP   BX
   POP   ES
   STI
   IRET

PUSH_REST_REGS: PUSH DX   ; FLAG WAS ON SO NEED
   PUSH DS                ; RE-SECHEDULE
   PUSH SI
   PUSH DI
   MOV AX,SEG GLOBALLOCK
   MOV  ES, AX
CK: MOV AL,119               ; LOCK GLOBAL LOCK
   LOCK XCHG ES:GLOBALLOCK,AL
   TEST AL,AL
   JNZ CK
   CALL RDYTHISVP
   MOV CX,77H             ; JUMP TO SCHEDULER
   JMP  INTJOIN

INTRET: POP DI
   POP SI                     ; RETURN FOR
   POP DS                     ; PROCESS WHICH
   POP DX                     ; HAD PREVIOUSLY
   POP CX                     ; BEEN INTERRUPTED
         ; UNLOCK GLOBAL$LOCK
   MOV  AX,SEG GLOBALLOCK
   MOV  ES, AX
   MOV  ES:GLOBALLOCK,Ø
   POP AX
   POP BX
   POP ES
   STI
   IRET

   INTERRUPT_HANDLER ENDP

;**************************************************************
;**************************************************************
;*    HARDWARE INTERRUPT FLAG                                 *
;*                                                            *

   HARDWARE_INT_FLAG  PROC  NEAR

   ASSUME CS:SCHEDULER
   ASSUME DS:NOTHING
   ASSUME SS:NOTHING
   ASSUME ES:NOTHING
HDW_FLAG: MOV  AX,SEG PRDS
   MOV  ES, AX
   MOV  BX,ØH
   MOV  CL,ES:PRDS[BX]     ;GET CPU #
   MOV  CH,Ø               ; RETURN IN BX
   MOV  BX,CX
   MOV  AX,SEG HDWINTFLAG  ;SET UP HDW$INT$FLAG
   MOV  ES, AX              ;      SEGMENT
   RET                      ; RETURN IN ES REG

HARDWARE_INT_FLAG  ENDP
SCHEDULER ENDS
END
```

# APPENDIX J

## GLOBAL DATA BASE AND INITIAL PROCESS CODE

Two files are contained in this appendix: GLOBAL.SRC AND INITK.SRC. They are separately compiled with the LARGE attribute. They are linked into the file: KORE.LNK. They are represented in the memory map for KORE presented at the end of Appendix F. INITK will be overwritten by an initialization module on each real processor.

```
/*****************************************************************/
/*****************************************************************/
/*0009************************************************************/
/* FILE:          GLOBAL1.SRC
   VERSION:       BREWER 3-18-84
   PROCEDURES
     DEFINED:   NONE

   REMARKS:THIS MODULE CONTAINS DECLARATIONS FOR ALL THE
           GLOBAL DATA THAT RESIDES IN SHARED COMMON
           MEMORY.  IT'S LOCATED THERE BY THE LOCATE COM-
           MAND AND BY SPECIFYING THAT THE
           GLOBAL$MODULE_DATA SEGMENT BE LOCATED AT SOME
           ABSOLUTE ADDRESS.
                                    */
/*****************************************************************/

GLOBAL$MODULE:  DO;

/*****************************************************************/
/*****************************************************************/
/*  THE FOLLOWING THREE LITERAL DECLARATIONS ARE ALSO       */
/*  GIVEN IN THE LEVEL1 & LEVEL2 MODULES OF THE OPERATING */
/*  SYSTEM.  A CHANGE HERE WOULD HAVE TO BE REFLECTED IN    */
/*  THOSE MODULES ALSO.                                      */

DECLARE
   MAX$CPU                    LITERALLY  '10',
   MAX$VPS$CPU                LITERALLY  '10',
   MAX$CPU$$$MAX$VPS$CPU LITERALLY '100';

DECLARE
   GLOBAL$LOCK BYTE PUBLIC INITIAL(0);

      /*  THIS SHOULD REFLECT THE MAX$CPU ABOVE */
DECLARE
   NR$RPS            BYTE PUBLIC INITIAL(0),
   NR$VPS(MAX$CPU) BYTE PUBLIC
                          INITIAL(0,0,0,0,0,0,0,0,0,0);

DECLARE HDW$INT$FLAG(MAX$CPU) BYTE PUBLIC;


DECLARE EVENTS  BYTE  PUBLIC INITIAL(1);

DECLARE LOCAL$CLUSTER$ADDR WORD PUBLIC;


DECLARE EVC$TBL(100) STRUCTURE
          (EVC$NAME      BYTE,
           VALUE         WORD,
           REMOTE$ADDR  WORD,
            THREAD        BYTE)  PUBLIC
                                 INITIAL(0FEH,0,0FFFFH,255);
        /* EVC "FE" IS RESERVED FOR THE OP SYS   */


DECLARE CPU$INIT BYTE PUBLIC INITIAL(0);

DECLARE SEQUENCERS      BYTE  PUBLIC INITIAL(0);

DECLARE SEQ$TABLE(100) STRUCTURE
           (SEQ$NAME       BYTE,
            SEQ$VALUE      WORD) PUBLIC;

DECLARE VPM( MAX$CPU$$$$MAX$VPS$CPU ) STRUCTURE
           (VP$ID                  BYTE,
            VP$STATE               BYTE,
            VP$PRIORITY            BYTE,
            EVC$THREAD             BYTE,
            EVC$AW$VALUE           WORD,
            SP$REG                 WORD,
            SS$REG                 WORD)  PUBLIC;


END; /* MODULE */

/*************************************************************/
```

```
/************************************************************************/
/*   INITK      MODULE                            BREWER 8-18-84    */
/*------------------------------------------------------------------*/
/* THE CODE SEGMENT OF THIS MODULE IS WHAT RESERVES SPACE */
/* BY THE OS FOR THE USER INITIAL PROCESS.  THIS IS       */
/* EXECUTABLE IN IT'S OWN RIGHT.  THUS IF THE USER DOES   */
/* NOT PROVIDE AN INITIAL PROCESS THIS ONE WILL EXECUTE,  */
/* BLOCK ITSELF, AND IDLE THE CPU.  THE ADDRESS OF THE    */
/* INITIAL CODE SEGMENT IS PROVIDED TO LEVEL1 AND IT IS   */
/* REFLECTED IN THE PLM LOCATE COMMAND.  THE ADDRESSES    */
/* PROVIDED MUST AGREE.  THIS PROCESS HAS THE HIGHEST     */
/* PRIORITY AND WILL ALWAYS BE SCHEDULED FIRST BY THE     */
/* SCHEDULER.                                             */
/*------------------------------------------------------------------*/
/* CALLS MADE TO:   AWAIT                                 */
/************************************************************************/
      INIT$MOD: DO;

/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*    DECLARE
/*     MSG13(*) BYTE INITIAL(10,'ENTERING INITIAL PROCESS ',
/*                                    13,10,'%');
/*    OUT$LINE: PROCEDURE( PTR ) EXTERNAL;
/*       DECLARE PTR POINTER;
/*    END;
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
AWAIT:  PROCEDURE( NAME, VALUE ) EXTERNAL;
          DECLARE NAME BYTE, VALUE WORD;
END;

INITIAL$PROC: PROCEDURE PUBLIC;

  DECLARE I BYTE;
/* AFTER INITIALIZATION THIS PROCESS BLOCKS      */
/* ITSELF TO ALLOW THE NEWLY CREATED PROCESSES   */
/* TO BE SCHEDULED.                              */
/* THIS AREA SHOULD BE WRITTEN OVER BY USER INIT */
/* PROCEDURE MODULE.                             */
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*    CALL OUT$LINE(@MSG13);
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/
/*** MXTRACE ***** MXTRACE ***** MXTRACE ***** MXTRACE ***/

    CALL AWAIT( 0FEH, 1);
  END;   /* INITIAL$PROC */
END;  /* INIT$MOD */
```

# APPENDIX K

## NI3010 DEVICE DRIVER AND PACKET PROCESSOR SOURCE CODE

This code consists of PL/I-86 modules and 8086 assembly language modules. PL/I-86 is primarily an applications programming language, rather than a systems development language. As such, it does not have the language features to gain access to the 8086 processor or MULTIBUS hardware. In situations where it is necessary to access hardware-dependent components, RASM86 [Ref. 18] modules are called. These assembly language routines are located in file ASMROUT.A86 (assembly language routines), and are linked with the PL/I-86 modules.

As described in detail in Chapter IV, the Driver is a MCORTEX *system* process with a dedicated real processor. Its linking conventions and use of MCORTEX primitives are identical to any user process. The notable exception is the use of its initialization module to define the cluster address, create sequencers, create eventcounts, and distribute the eventcounts.

The Driver also reads a file called ADDRESS.DAT to determine its own physical Ethernet address and addresses to load into its multicast (or group) address table. Note the type of data in ADDRESS.DAT must be bit string for addresses, and fixed binary for the number of group addresses.

The SYSINIT1.PLI file is the initialization module for Cluster 1 and SYSINIT2.PLI is the initialization module for Cluster 2. These files and ADDRESS.DAT are the only system files that must be changed when new MCORTEX processes are added, causing a change in eventcount distributivity. MCORTEX processes may be readily ported in executable image form from one cluster to another. The eventcount distribution changes only require a change in the Driver initialization modules and the cluster ADDRESS.DAT files. The amount of recompilation and linking is kept to the absolute minimum with this schema.

The contents of SYSDEF.PLI (Appendix E), ADDRESS.DAT, and the Driver initialization modules reflect the current system configuration. This is the demonstration process described in Appendix F.

Due to thesis format requirements, the structure of the source code is slightly altered, i.e., PL/I statements are not necessarily compilable as illustrated.

```
*****************************************************************
*****************************************************************
***         Cluster 1 ADDRESS.DAT file                        ***
*****************************************************************

1,
'00000000'b,'00000001'b,
'00000000'b,'00000001'b

*****************************************************************
*****************************************************************
***               SYSINIT1.PLI file                           ***
*****************************************************************


sysinit1: proc options (main);

     %include 'sysdef.pli';

     %replace

        EVC_TYPE          by '00'b4;

     /* main */

        call define_cluster ('0001'b4); /* must be called
                                           prior to creating
                                           evc's  */

        /**** USER ****/

        call create_evc (TRACK_IN);
        call create_evc (TRACK_OUT);
        call create_evc (MISSILE_ORDER_IN);
        call create_evc (MISSILE_ORDER_OUT);

        /*** SYSTEM ***/


        call create_evc (ERB_READ);
        call create_evc (ERB_WRITE);
        call create_seq (ERB_WRITE_REQUEST);

        /* distrib. map called after eventcounts have
           been created */

        call distribution_map (EVC_TYPE, TRACK_IN,'0003'b4);
          /* local and remote copy of TRACK_IN needed */
        call distribution_map (EVC_TYPE, MISSILE_ORDER_OUT,
                               '0003'b4);
        call create_proc ('fc'b4, '80'b4,
                          '0950'b4, '0800'b4, '005f'b4,
                           '0439'b4, '0800'b4, '0800'b4);
     call await ('fe'b4, '01'b4);

end sysinit1;

************************************************************
************************************************************
***          Cluster 2 ADDRESS.DAT file                  ***
************************************************************

1,
'00000000'b,'00000010'b,
'00000000'b,'00000010'b

************************************************************
************************************************************
***                     SYSINIT2.PLI file                ***
************************************************************

sysinit2: proc options (main);

     %include 'sysdef.pli';

     %replace

        EVC_TYPE          by '00'b4;

     /* main */

        call define_cluster ('0002'b4); /* must be called
                                           prior to creating
                                           evc's  */

        /**** USER ****/

        call create_evc (TRACK_IN);
        call create_evc (TRACK_OUT);
        call create_evc (MISSILE_ORDER_IN);
        call create_evc (MISSILE_ORDER_OUT);

        /*** SYSTEM ***/


        call create_evc (ERB_READ);
        call create_evc (ERB_WRITE);
        call create_seq (ERB_WRITE_REQUEST);
        /* distrib. map called after eventcounts have
           been created */

        call distribution_map (EVC_TYPE,TRACK_OUT,'0003'b4);
          /* local and remote copy of TRACK_IN needed */
        call distribution_map (EVC_TYPE, MISSILE_ORDER_IN,
                                     '0003'b4);
      /* local and remote copy of MISSILE_ORDER_IN needed */
        call create_proc ('fc'b4, '80'b4,
                               '0950'b4, '0800'b4, '005f'b4,
                               '0439'b4, '0800'b4, '0800'b4);
        call await ('fe'b4, '0001'b4);

end sysinit2;

***************************************************************
***************************************************************
***                   NI3010.DCL file                        ***
***************************************************************

   %replace

/*                     I/O port addresses

These values are specific to the use of the INTERLAN
NI3010 MULTIBUS to ETHERNET interface board. Any change
to the I/O port address of '00b0' hex (done so with a DIP
switch) will require a change to these addresses to
reflect that change.
                                                 */



   command_register                 by 'b0'b4,
   command_status_register          by 'b1'b4,
   transmit_data_register           by 'b2'b4,
   interrupt_status_reg             by 'b5'b4,
   interrupt_enable_register        by 'b8'b4,
   high_byte_count_reg              by 'bc'b4,
   low_byte_count_reg               by 'bd'b4,


   /*  end of I/O port addresses */

   /*  Interrupt enable status register values */
   disable_ni3010_interrupts        by '00'b4,
   ni3010_intrpts_disabled          by '00'b4,
   receive_block_available          by '04'b4,
   transmit_dma_done                by '06'b4,
   receive_dma_done                 by '07'b4,
   /*   end register values */

   /*   Command Function Codes */

   module_interface_loopback      by '01'b4,
   internal_loopback              by '02'b4,
   clear_loopback                 by '03'b4,
   go_offline                     by '08'b4,
   go_online                      by '09'b4,
   onboard_diagnostic             by '0a'b4,
   clr_insert_source              by '0e'b4,
   load_transmit_data             by '28'b4,
   load_and_send                  by '29'b4,
   load_group_addresses           by '2a'b4,
   reset                          by '3f'b4;

/*   end Command Function Codes          */

**************************************************************
**************************************************************
***                  SYSDEV.PLI file                        ***
**************************************************************

sysdev: procedure;


/* Date:             1 SEPTEMBER 1984

   Programmer:       David J. BREWER

   Module Function:  To serve as the Ethernet Communication
                     Controller Board (NI3010) device
                     handler. This process is scheduled
                     under MCORTEX and consumes Ethernet
                     Requests Packets (ERP) generated by
                     the SYSTEM$IO routine in LEVEL2.SRC.
                     It also processes any inbound packets
                     by analyzing the packet contents and
                     making the appropriate MCORTEX calls.

                                                          */



   %replace

      evc_type            by '00'b4,
      erb_block_len          by 20,
      erb_block_len_m1       by 19,
      infinity            by 32767;

   %include 'sysdef.pli';
```

```
DECLARE

   1    erb(0:erb_block_len_m1) based (block ptr),
           2 command      bit (8),
           2 type_name    bit (8),
           2 name_value   bit (16),
           2 remote_addr  bit (16);

DECLARE

   1    transmit_data_block based (xmit_ptr),

           2 destination_address_a
                 bit (8) ,
           2 destination_address_b
                 bit (8),
           2 destination_address_c
                 bit (8),
           2 destination_address_d
                 bit (8),
           2 destination_address_e
                 bit (8) ,
           2 destination_address_f
                 bit (8) ,
           2 source_address_a
                 bit (8) ,
           2 source_address_b
                 bit (8),
           2 source_address_c
                 bit (8),
           2 source_address_d
                 bit (8),
           2 source_address_e
                 bit (8) ,
           2 source_address_f
                 bit (8) ,
           2 type_field_a
                 bit (8) ,
           2 type_field_b
                 bit (8),
           2 data (46) bit (8),


   1 receive_data_block based (rcv_ptr),

      2 frame_status           bit (8),
      2 null_byte              bit (8) ,
      2 frame_length_lsb       bit (8) ,
      2 frame_length_msb       bit (8) ,
          2 destination_address_a   bit (8) ,
          2 destination_address_b   bit (8) ,
          2 destination_address_c   bit (8) ,
          2 destination_address_d   bit (8) ,
          2 destination_address_e   bit (8) ,
          2 destination_address_f   bit (8) ,
          2 source_address_a        bit (8) ,
          2 source_address_b        bit (8) ,
          2 source_address_c        bit (8) ,
          2 source_address_d        bit (8) ,
          2 source_address_e        bit (8) ,
          2 source_address_f        bit (8) ,
          2 type_field_a            bit (8) ,
          2 type_field_b            bit (8) ,
          2 data(46)                bit (8) ,
          2 crc_msb                 bit (8) ,
          2 crc_upper_middle_byte   bit (8) ,
          2 crc_lower_middle_byte   bit (8) ,
          2 crc_lsb                 bit (8) ,

       (xmit_ptr, rcv_ptr,block_ptr) pointer,
       index fixed bin (15),
       (addr_e, addr_f) bit (8),
       address file,
       copy_ie_register bit (8),
       (cluster_addr,erb_write_value,i) bit (16),
       (j,k) fixed bin (15),
       reg_value bit (8) ,
       write_io_port entry (bit (8), bit (8)),
       read_io_port  entry (bit (8), bit (8)),
       initialize_cpu_interrupts     entry,
       enable_cpu_interrupts         entry,
       disable_cpu_interrupts        entry,
       write_bar entry (bit(16));

       /*   end module listing  */


    %replace

    /*  codes specific to the Intel 8259a Programmable
        Interrupt Controller (PIC)     */

                       icw1_port_address        by 'c0'b4,
 /* note that */       icw2_port_address        by 'c2'b4,
 /* icw2,icw4,*/       icw4_port_address        by 'c2'b4,
 /* and ocw   */       ocw_port_address         by 'c2'b4,
 /* use same  */
 /* port addr */

       /* note: icw ==> initialization
                        control
```

```
                    word

                    ocw ==> operational
                    command
                    word                */


                    icw1                         by '13'b4,

                 /* single PIC configuration, edge
                    triggered input              */

                    icw2                         by '40'b4,

                 /* most significant bits of vectoring
                    byte: for an interrupt 5,
                    the effective address will be
                    (icw2 + interrupt #) * 4 which
                    will be (40 hex + 5) * 4 =
                    114 hex                        */

                    icw4                        by '0f'b4,

                 /* automatic end of interrupt
                    and buffered mode/master       */

                    ocw1                        by '8f'b4;

               /* unmask interrupt 4 (bit 4),      */
               /* interrupt 5 (bit 5), and         */
               /* interrupt 6 (bit 6),mask all others */

                         /* end 8259a codes */


        /* include constants specific to the NI3010
        board                                    */

    %include 'ni3010.dcl';
```

```
/*************************************************************************/

                    /*  Main Pody */


call write_io_port(interrupt_enable_register,
                   disable_ni3010_interrupts);
call initialize_pic;
call initialize_cpu_interrupts;
call read_io_port (command_status_register,reg_value);
call perform_command (reset);

call program_group_addresses;
/* assignments to the source and destination address
   fields that will not change */

call perform_command (clr_insert_source);
/* NI3010 performance is enhanced in this mode */

unspec(block_ptr) = block_ptr_value;
unspec(rcv_ptr) =  rcv_ptr_value;
unspec(xmit_ptr) = xmit_ptr_value;

/* make one time assignments to transmit data block */

transmit_data_block.destination_address_a = '03'b4;
transmit_data_block.destination_address_b = '00'b4;
transmit_data_block.destination_address_c = '00'b4;
transmit_data_block.destination_address_d = '00'b4;
transmit_data_block.source_address_a = '03'b4;
transmit_data_block.source_address_b = '00'b4;
transmit_data_block.source_address_c = '00'b4;
transmit_data_block.source_address_d = '00'b4;

/* get the local cluster address - file was
   opened in proc program_group_addresses   */


get file (address) list (addr_e, addr_f);
transmit_data_block.source_address_e = addr_e;
transmit_data_block.source_address_f = addr_f;

cluster_addr = addr_e || addr_f;
put skip (2) edit ('*** CLUSTER ',cluster_addr,
                   ' Initialization Complete ***')
                   (col(15),a,b4(4),a);
i = '0001'b4;
call perform_command (go_online);

/* at this point copy_ie_reg = RBA , but
   ie_reg on NI3010 is actually disabled */
call disable_cpu_interrupts;
```

```
do k = 1 to infinity;
  /* note: interrupt not allowed during a
     call to MCORTEX primitive       */

   erb_write_value = read(ERB_WRITE);
       /* In the MXTRACE version of the RTOS
          all primitive calls clear    and
          set interrupts (diagnostic message
          routines), so the NI3010 interrupts
          must be disabled on entry to MXTRACE */
   do while (erb_write_value < i);
             /* busy waiting */
      erb_write_value = read(ERB_WRITE);
      copy_ie_register=receive_block_available;
      call write_io_port(interrupt_enable_register,
                         receive_block_available);
      call enable_cpu_interrupts;
      /* if a packet has been received,this
         is when an interrupt may occur - can
         see that outbound packets are always
         favored.     */
      do j = 1 to 1000;
        /* interrupt window for packets received */
      end; /* do j */
      call disable_cpu_interrupts;
      if (copy_ie_register = receive_dma_done) then
      do;
       /* receive DMA operation started, so let
          finish. */
           call enable_cpu_interrupts;
           do while (copy_ie_register=receive_dma_done);
           end;
           call disable_cpu_interrupts;
      end; /* ift */
      copy_ie_register = disable_ni3010_interrupts;
      call write_io_port(interrupt_enable_register,
                         disable_ni3010_interrupts);
   end; /* busy */

   /* ERB has an ERP in it, so process it */
   /* no external interrupts (RBA) until
      the ERP is consumed and the packet
      gets sent       */
   index = mod((fixed(i) - 1), erb_block_len);
       /* 32k limit on parameter to fixed fcn. */
   transmit_data_block.data(1) = erb(index).command;
   transmit_data_block.data(2) = erb(index).type_name;
   transmit_data_block.data(3) =
                           substr(erb(index).name_value,
                           9,8);
      transmit_data_block.data(4) =
                              substr(erb(index).name_value,
                              1,8);
      transmit_data_block.destination_address_e =
                    substr(erb(index).remote_addr, 1,8);
      transmit_data_block.destination_address_f =
                    substr(erb(index).remote_addr, 9,8);

      call advance (ERB_READ); /* caution here !!!!
                              an ADVANCE will result in a
                              call to VP$SCHEDULER, which
                              will set CPU interrupts on exit.
                              It's the reason NI3010 interrupts
                              are disabled first in the
                              Do While loop above.  */

   /* packet ready to go, so send it */

   call transmit_packet;
   /* copy_ie_register = RBA , but not actual register */
   call disable_cpu_interrupts;

   /* setting up for next ERP consumption */
   i = add2bit16(i, '0001'b4);

end;  /* do forever */

        /* end main body */

/****************************************************************/



initialize_pic:   procedure;


   DECLARE

      write_io_port entry (bit (8) , bit(8));

   call write_io_port (icw1_port_address,icw1);
   call write_io_port (icw2_port_address,icw2);
   call write_io_port (icw4_port_address,icw4);
   call write_io_port (ocw_port_address,ocw1);

end initialize_pic;

/****************************************************************/
```

```
perform_command:        procedure (command);

   DECLARE
      command bit (8) ,
      reg_value bit (8) ,
      srf bit (8) ,
      write_io_port entry (bit (8), bit (8) ),
      read_io_port  entry (bit (8), bit (8) );

   /* end declarations */

   srf = 'Ø'b4;
   call write_io_port (command_register,command);
   do while ((srf & 'Ø1'b4) = 'ØØ'b4);
      call read_io_port (interrupt_status_reg, srf);
   end;  /* do while */
   call read_io_port (command_status_register, reg_value);
   if (reg_value > 'Ø1'b4) then
   do;
      /* not (SUCCESS or SUCCESS with Retries) */

      put skip edit ('*** ETHERNET  Board Failure ***')
                    (col(2Ø),a);
                /* when this occurs, run the diagnostic
                   routine T3Ø1Ø/Cx, where x is the
                   current cluster number */
      stop;
   end; /* itd */


end perform_command;





/*************************************************************/


transmit_packet: procedure  external;



   DECLARE
      srf bit (8) ,
      reg_value bit (8) ,
      write_io_port entry (bit (8) , bit (8) ),
      read_io_port entry  (bit (8) , bit (8) ),
      enable_cpu_interrupts         entry,
      disable_cpu_interrupts        entry,
      write_bar entry (bit(16));
```

```
    /* begin */

  srf = '0'b4;
  call write_bar (xmit_ptr_value);
  call write_io_port(high_byte_count_reg,'00'b4);
  call write_io_port(low_byte_count_reg,'3c'b4);
  copy_ie_register = transmit_dma_done;
  call write_io_port(interrupt_enable_register,
                         transmit_dma_done);
  call enable_cpu_interrupts;
  do while (copy_ie_register = transmit_dma_done);
  end;    /* loop until the interrupt handler
             takes care of the TDD interrupt -
             it sets copy_ie_register = RBA */
  call perform_command (load_and_send);

end transmit_packet;

/*****************************************************************/


HL_interrupt_handler: procedure external;



  /* This routine is called from the low level
     8086 assembly language interrupt routine */

  DECLARE

     write_io_port entry (bit (8) , bit (8) ),
     read_io_port entry (bit (8) , bit (8) ),
     enable_cpu_interrupts          entry,
     disable_cpu_interrupts         entry,
     write_bar entry (bit(16));

  /*  begin   */

  call write_io_port(interrupt_enable_register,
                      disable_ni3010_interrupts);

  if (copy_ie_register = receive_block_available)
  then do;

     call write_bar (rcv_ptr_value);
     call write_io_port(high_byte_count_reg,'05'b4);
     call write_io_port(low_byte_count_reg,'f2'b4);

     /* initiate receive DMA */

     copy_ie_register = receive_dma_done;
     call write_io_port(interrupt_enable_register,
                              receive_dma_done);

   end;    /* do */
   else
      if (copy_ie_register = receive_dma_done) then
      do;
         call process_packet;
         copy_ie_register = receive_block_available;
         call write_io_port(interrupt_enable_register,
                            receive_block_available);
      end;  /* if then do */
      else
         if (copy_ie_register = transmit_dma_done)
         then do;

              copy_ie_register = receive_block_available;
              /* NI3010 interrupts disabled on entry */
         end;   /* if then do */

               end HL_interrupt_handler;


/*************************************************************/

process_packet: procedure;

DECLARE

   local_evc_value bit (16),
   data_ptr pointer,
   remote_evc_value bit (16) based (data_ptr);

   if (receive_data_block.data(1) = evc_type) then
   do;
      data_ptr = addr(receive_data_block.data(3));

      /* remote_evc_value now has a value */

      local_evc_value = read(receive_data_block.data(2));

      do while (local_evc_value < remote_evc_value);

         call advance (receive_data_block.data(2));
         local_evc_value = add2bit16(local_evc_value,
                                     '0001'b4);

      end;
      call disable_cpu_interrupts;
      /* this must be done due to setting of
         cpu interrupts by calls to MCORTEX's
         VP$SCHEDULER via ADVANCE */
   end; /* itd */
      /* only type packet in this limited implem. */

end process_packet;

/*****************************************************************/

program_group_addresses: procedure;


   DECLARE


      1  group_addr(40) based (group_ptr),

         2 mc_group_field_a
              bit (8),
         2 mc_group_field_b
              bit (8),
         2 mc_group_field_c
              bit (8),
         2 mc_group_field_d
              bit (8),
         2 mc_group_field_e
              bit (8),
         2 mc_group_field_f
              bit (8);


   DECLARE

      (group_ptr,p) pointer,
      (field_e, field_f) bit (8),
      bit_8_groups bit (8) based (p),
      (i,num_groups,groups_times_6) fixed bin (7);

   unspec(group_ptr) = xmit_ptr_value;
   open file (address) stream input;
   get file (address) list (num_groups);
   do i = 1 to num_groups;

      group_addr(i).mc_group_field_a = '03'b4;
      group_addr(i).mc_group_field_b = '00'b4;
      group_addr(i).mc_group_field_c = '00'b4;
      group_addr(i).mc_group_field_d = '00'b4;
      get file (address) list (field_e,field_f);
      group_addr(i).mc_group_field_e = field_e;
      group_addr(i).mc_group_field_f = field_f;

   end;    /* do i */

   call disable_cpu_interrupts;
   call write_bar (xmit_ptr_value);
   call write_io_port(high_byte_count_reg, '00'b4);
   groups_times_6 = 6 * num_groups;
   p = addr (groups_times_6);
   call write_io_port(low_byte_count_reg, bit_8_groups);
   copy_ie_register = transmit_dma_done;
   call write_io_port(interrupt_enable_register,
                          transmit_dma_done);
   call enable_cpu_interrupts;
   do while (copy_ie_register = transmit_dma_done);
   end;   /* loop until the interrupt handler
             takes care of the TDD interrupt -
             it sets COPY_IE_REG = RBA  */

   call perform_command(load_group_addresses);

end program_group_addresses;

/*****************************************************************/


end;   /* system device handler and packet processor */
```

```
*******************************************************************
*******************************************************************
***                  ASMROUT.A86 file                          ***
*******************************************************************

extrn hl_interrupt_handler : far

public write_io_port
public read_io_port
public write_bar
public initialize_cpu_interrupts
public enable_cpu_interrupts
public disable_cpu_interrupts
;**********************************************************************


write_io_port:


     ; Parameter Passing Specification:

     ;                      entry                      exit
     ;
     ; parameter 1     <port address>              <unchanged>
     ;
     ; parameter 2     <value to be outputted>  <unchanged>
     ;
     ;

          dseg

          port_address    rb    1

          cseg


          push bx! push si! push dx! push ax
          mov  si, [bx]
          mov  al, [si]
          mov  port_address, al
          mov  si, 2[bx]
          mov  al, [si]
          mov  dl, port_address
          mov  dh, 00h
          out  dx, al
          pop ax! pop dx! pop si! pop bx
          ret

;*******************************************************************
```

```
read_io_port:


    ; Parameter Passing Specification
    ;
    ;                       entry                     exit
    ;
    ; parameter 1     <port address>             <unchanged>
    ; parameter 2     <meaningless>           <register value>

          cseg
          push bx! push si! push dx! push ax
          mov  si,  [bx]
          mov  al,  [si]
          mov  port_address,  al
          mov  si, 2[bx]
          mov  dl, port_address
          mov  dh,  00h
          in   al,  dx
          mov  [si], al
          pop  ax! pop dx! pop si! pop bx!
          ret
;**************************************************************************

write_bar:


   ; Parameter Passing Specification
   ;
   ; parameter 1 (and only): the address of the data block to be
   ;                         transmitted or received.

          dseg

          e_bar_port      equ  0b9h
          h_bar_port      equ  0bah
          l_bar_port      equ  0bbh
          temp_e_byte     rb   1
          temp_es         rw   1

          cseg

   ; This module computes a 24 bit address from a 32 bit
   ; address - actually it's a combination of the ES register
   ; and the IP passed via a parameter list.

          push bx! push ax! push cx! push es! push dx! push si


          mov  dx,  0800h         ; shared memory segment
          mov  es,  dx
          mov   temp_es, es
          mov   dx,  es
          mov   si,  [bx]
          mov   ax,  [si]
          mov   cl,  12
          shr   dx,  cl
          mov   temp_e_byte,   dl
          mov   dx,  temp_es
          mov   cl,  4
          shl   dx,  cl
          add   ax,  dx
          jnc   no_add
add_1:    inc   temp_e_byte
no_add:   out   l_bar_port, al
          mov   al, ah
          out   h_bar_port, al
          mov   al, temp_e_byte
          out   e_bar_port, al
          pop si! pop dx! pop es! pop cx! pop ax! pop bx
          ret

;-----------------------------------------------------


initialize_cpu_interrupts:


    ; Module Interface Specification:

    ;     Caller:         Ethertest(PL/I) Procedure

    ;     Parameters:     NONE

    initmodule cseg common
               org 114h
               int5_offset   rw 1
               int5_segment  rw 1

               cseg
               push bx
               push ax
               mov  bx,  offset interrupt_handler
               mov  ax,  Ø
               push ds
               mov  ds,  ax
               mov  ds:int5_offset, bx
               mov  bx, cs
               mov  ds:int5_segment, bx
               pop  ds
               pop  ax
               pop  bx
               sti
```

```
                ret

;---------------------------------------------------------------

enable_cpu_interrupts:

     ; Module Interface Specification:

     ;     Caller:          Ethertest(PL/I) Procedure

     ;     Parameters:      NONE

                sti
                ret

;---------------------------------------------------------------

disable_cpu_interrupts:

     ; Module Interface Specification:

     ;     Caller:          Ethertest(PL/I) Procedure
     ;     Parameters:      none

                cli
                ret

;---------------------------------------------------------------

interrupt_handler:


                ; IP, CS, and flags are already on stack
                ; save all other registers

                push ax
                push bx
                push cx
                push dx
                push si
                push di
                push bp
                push ds
                push es
                call hl_interrupt_handler ; high level source
                                          ; routine
                ; restore registers

                pop es
                pop ds
                pop bp
                pop di
                pop si
                pop dx
                pop cx
                pop bx
                pop ax
                sti
                iret

end
```

# APPENDIX L

## NI3010 DIAGNOSTIC CODE

In the event of an Ethernet board failure indication by the NI3010 Driver, the full range of NI3010 operations can be tested with this routine. Any changes to the port addresses of the NI3010 will have to be reflected in the NI3010.DCL file contained in Appendix K. This code will also have to be recompiled and relinked.

This routine is invoked with the CP/M-86 transient command: T3010/Cx, where x is the cluster to be tested. For example, T3010/C1 tests the NI3010 at Cluster 1. This diagnostic routine uses the factory default Ethernet physical address, so the boards should not be swapped between clusters without taking note of its physical address. The NI3010 Driver does not have this restriction. The file ASMROUT.A86 is linked with the module to allow access to hardware port addresses and to allow a low level assembly language interrupt handler to call a PL/I-86 interrupt handler. The LINK86 input option files are also included in this appendix.

```
************************************************************************
************************************************************************
**        T3010/C1.INP LINK86 input option file                      ***
************************************************************************

t3010/c1=
boardtst[code[ab[439]],data[ab[800],m[0],ad[82]],map[all]],
asmrout

************************************************************************
************************************************************************
**        T3010/C2.INP LINK86 input option file                      ***
************************************************************************

t3010/c2=
boardtst[code[ab[439]],data[ab[800],m[0],ad[82]],map[all]],
asmrout

************************************************************************
************************************************************************
***                TEST3010.DAT file                                 ***
************************************************************************

This is a highly reliable packet switching implementation!

************************************************************************
************************************************************************
***                NI3010 DIAGNOSTIC ROUTINE                         ***
************************************************************************

boardtst:     procedure options (main);


/*    Date:                 14 FEB 1984

      Programmer:           David J. Brewer

      Module  Function:   This  module,   and  associated
        submodules, are designed to fully diagnose the
        NI3010   Multibus   to  Ethernet   Commmunications
        Controller. If at any time, during the development
        of  software or hardware by a user/implementor  of
        ECCB   software a   fault is     suspected,    this
        comprehensive  diagnostic routine can be  executed
        under  CP/M  - 86 by invoking the  command  module
        (i.e.,  transient  command) 'T3010/Cx', where x
        represents the cluster location.

                                              */

   DECLARE
```

```
        1    transmit_data_block based(trans_blk_ptr),

                               2 destination_address_a
/*             ---->*/               fixed bin (7),
/*  assigned   |    */ 2 destination_address_b
/*     by      ---->*/               fixed bin (7) ,
/*   XEROX     !    */ 2 destination_address_c
/*             ---->*/               fixed bin (7),
                               2 destination_address_d
/*             ---->*/               fixed bin (7),
/*  assigned   |    */ 2 destination_address_e
/*     by      ---->*/               fixed bin (7),
/*  INTERLAN   !    */ 2 destination_address_f
                                     fixed bin (7),
                               2 type_field_a
                                     fixed bin (7),
                               2 type_field_b
                                     fixed bin (7),
                               2 data_bytes (1500)
                                     char (1),


        1 receive_data_block based (rec_blk_ptr),

                   2 frame_status               bit(8),
                   2 null_byte                  fixed bin (7),
                   2 frame_length_lsb           fixed bin (7),
                   2 frame_length_msb           fixed bin (7),
                   2 destination_address_a      fixed bin (7),
                   2 destination_address_b      fixed bin (7),
                   2 destination_address_c      fixed bin (7),
                   2 destination_address_d      fixed bin (7),
                   2 destination_address_e      fixed bin (7),
                   2 destination_address_f      fixed bin (7),
                   2 source_address_a           fixed bin (7),
                   2 source_address_b           fixed bin (7),
                   2 source_address_c           fixed bin (7),
                   2 source_address_d           fixed bin (7),
                   2 source_address_e           fixed bin (7),
                   2 source_address_f           fixed bin (7),
                   2 type_field_a               fixed bin (7),
                   2 type_field_b               fixed bin (7),
                   2 data_bytes (1500)          char (1),
                   2 crc_msb                    fixed bin (7),
                   2 crc_upper_middle_byte      fixed bin (7),
                   2 crc_lower_middle_byte      fixed bin (7),
                   2 crc_lsb                    fixed bin (7),


        test3010 file,
        copy_ie_register fixed bin (7),
        copy_command_status_register fixed bin (7),
        (i,j,k) fixed bin (15),
        reg_value fixed bin (7),
        operation fixed bin (7),
        cluster fixed bin (7),
        border (80) char (1) static initial ((80)'-'),
        (trans_blk_ptr,rec_blk_ptr) pointer,


    /*   Modules external to this module */

        write_io_port entry (fixed bin (7),fixed bin (7)),
        read_io_port  entry (fixed bin (7),fixed bin (7)),
        initialize_cpu_interrupts     entry,
        enable_cpu_interrupts         entry,
        disable_cpu_interrupts        entry,
        write_bar entry (pointer);

        /*   end module listing  */


   %replace

    /*  codes specific to the Intel 8259a Programmable
        Interrupt Controller (PIC)      */

                  icw1_port_address          by 'c0'b4,
/* note that */  icw2_port_address          by 'c2'b4,
/* icw2,icw4,*/  icw4_port_address          by 'c2'b4,
/* and ocw   */  ocw_port_address           by 'c2'b4,
/* use same  */
/* port addr */

        /* note: icw ==> initialization
                         control
                         word

                 ocw ==> operational
                         command
                         word                */


        icw1                        by '13'b4,

        /* single PIC configuration, edge
           triggered input          */

        icw2                        by '40'b4,
        /* most significant bits of vectoring
           byte: for an interrupt 5,
           the effective address will be
           (icw2 + interrupt #) * 4 which
```

```
          will be (40 hex + 5) * 4 =
          114 hex                 */

       icw4                       by '0f'b4,

       /* automatic end of interrupt
          and buffered mode/master    */

       ocw1                       by '9f'b4,

       /* unmask interrupt 5 (bit 5) and
          interrupt 6. mask all others   */

       /* end 8259a codes */


       cluster1                             by 1,
       cluster2                             by 2,
       packet_received                      by 1,
       await_packet                         by 0;

   /* include constants specific to the NI3010
      board                                          */



   %include 'ni3010.dcl';




/*************************************************************/

       /*  Main Body */


   cluster = cluster2;
      /* conditional to set up own address for loopbacks */
   put list ('^z'); /* clear screen */
   put skip;
   put edit ((border (i) do i = 1 to 80)) (a);
   put skip (2) edit ('NI3010 Diagnostic Routine')
                    (col(20),a)           ;
   put skip (2);
   put skip edit ('Command Issued','Result') (col(5),a,
                 col(50),a);
   put edit ('**************','******') (col(5),a,
             col(50),a);
   put skip (2);

   unspec(trans_blk_ptr) = '8000'b4;
   unspec(rec_blk_ptr)   = '8600'b4;
```

```
/* with a DS register value of 0800h in the link
   command, this will place packets in extended
   memory (therefore DMA operation can take place */
transmit_data_block.destination_address_a = 2;
transmit_data_block.destination_address_b = 7;
transmit_data_block.destination_address_c = 1;
transmit_data_block.destination_address_d = 0;
if (cluster = cluster1) then
do;
   transmit_data_block.destination_address_e = 3;
   transmit_data_block.destination_address_f = -22;
      /* corresponds to 03-EA */
end;
else
do;  /* it's cluster 2 */
   transmit_data_block.destination_address_e = 4;
   transmit_data_block.destination_address_f = 10;
     /* corresponds to 04-0A */
end;
transmit_data_block.type_field_a = 0;
transmit_data_block.type_field_b = 0;
do i = 1 to 1500;
      transmit_data_block.data_bytes(i) = ' ';
end;


call read_io_port (command_status_register,reg_value);
call fill_data_block;
call initialize_pic;
call initialize_cpu_interrupts;
put skip edit ('Run Onboard Diagnostic') (col(5),a);
call perform_command (onboard_diagnostic);
put skip edit ('Perform  Module Interface Loopback')
              (col(5),a);
call perform_loopback (module_interface_loopback);
do i = 1 to 1500;
   receive_data_block.data_bytes (i) = ' ';
end;  /* do i */
put skip edit ('Perform Internal Loopback') (col(5),a);
call perform_loopback (internal_loopback) ;
do i = 1 to 1500;
   receive_data_block.data_bytes (i) = ' ';
end;  /* do i */
put skip edit ('Perform External Loopback') (col(5),a);
call perform_loopback (go_online); /* external loopback */
put skip (2);
put edit ((border (i) do i = 1 to 80)) (a);
put skip (2);

call perform_command(reset);
```

```
   /* end main body */

/***************************************************************/

   /* procedures    */


fill_data_block: procedure;



   DECLARE


      i fixed bin (15) static initial (1),
      end_of_file bit (1) static init ('0'b);

      /* begin */
      open file (test3010);
      on endfile (test3010)
         begin;
              end_of_file = '1'b;
         end;
      do while ( end_of_file = 0);
  get file(test3010)edit(transmit_data_block.data_bytes(i))
                          (a(1));
       i = i + 1;
      end; /* do while */

end;    /*  fill_data_block  */

    /***********************************************************/


initialize_pic:   procedure;


   DECLARE

      write io_port entry (fixed bin (7),fixed bin(7));
      call write_io_port (icw1_port_address,icw1);
      call write_io_port (icw2_port_address,icw2);
      call write_io_port (icw4_port_address,icw4);
      call write_io_port (ocw_port_address,ocw1);

end initialize_pic;

/***************************************************************/


perform_command:        procedure (command);
```

```
   DECLARE
      command fixed bin (7),
      reg_value fixed bin (7),
      srf fixed bin (7),
      write_io_port entry (fixed bin (7), fixed bin (7)),
      read_io_port  entry (fixed bin (7), fixed bin (7)),
      command_status_codes entry (fixed bin (7))
                           returns (char(30) varying);

   /* end declarations */

   srf = 0;
   call write_io_port (command_register,command);
   do while (mod(srf,2) = 0);
      call read io_port (interrupt_status_reg, srf);
   end;  /* do while */
   call read_io_port (command_status_register, reg_value);
   if (command ~= reset) then
   do;
      if (command ~= onboard_diagnostic) then
              put edit (command_status_codes(reg_value))
                       (col(50),a);
      else
              put edit (diagnostic_codes(reg_value))
                       (col(50),a);
   end;

                    end perform_command;

/****************************************************************/

perform_loopback:      procedure (command);

   DECLARE


      write_io_port entry (fixed bin (7), fixed bin (7)),
      read_io_port entry  (fixed bin (7), fixed bin (7)),
      initialize_cpu_interrupts     entry,
      enable_cpu_interrupts         entry,
      disable_cpu_interrupts        entry,
      write_bar entry (pointer),


      command_status_codes entry (fixed bin (7))
                           returns (char(30) varying),
      command fixed bin (7),
      status_code fixed bin (7),
      ie_reg_value fixed bin (7),
      srf fixed bin (7);

   /* end declare */
   operation = await_packet;
   srf = 0;
   call disable_cpu_interrupts;
   copy_ie_register = receive_block_available;
   call write_io_port (interrupt_enable_register,
                       receive_block_available);
   call enable_cpu_interrupts;
   call write_io_port (command_register,command);

   do while (mod(srf,2) = 0);
      call read_io_port (interrupt_status_reg, srf);
   end;    /* do while */

   /* status is available, so read it */

   call read_io_port (command_status_register, status_code);
   put edit (command_status_codes(status_code)) (col(50),a);
   call transmit_packet (transmit_data_block);

   do while (operation = await_packet);
      /* handler will change */
   end;


end perform_loopback;


/*****************************************************************/


transmit_packet: procedure (packet) external;



   DECLARE

      srf fixed bin (7),
      reg_value fixed bin (7),
      write_io_port entry (fixed bin (7), fixed bin (7)),
      read_io_port entry  (fixed bin (7), fixed bin (7)),
      enable_cpu_interrupts          entry,
      disable_cpu_interrupts         entry,
      write_bar entry (pointer),


            1    packet,

                   2 destination_address_a
/*          ---->*/         fixed bin (7),
/*  assigned  |    */ 2 destination_address_b
/*      by       ---->*/           fixed bin (7),
/*   XEROX      !     */ 2 destination_address_c
/*               ---->*/           fixed bin (7),
                        2 destination_address_d
/*               ---->*/           fixed bin (7),
/*  assigned    !     */ 2 destination_address_e
/*      by       ---->*/           fixed bin (7),
/*  INTERLAN    !     */ 2 destination_address_f
/*               ---->*/           fixed bin (7),
                        2 type_field_a
                                    fixed bin (7),
                        2 type_field_b
                                    fixed bin (7),
                        2 data_bytes (1500) char (1);

      /* begin */
   srf = 0;
   call write_io_port(interrupt_enable_register,
                        disable_ni3010_interrupts);
   call write_bar (addr(packet));
   call write_io_port(high_byte_count_reg, 5); /* 1508 */
                                                /* bytes*/
   call write_io_port(low_byte_count_reg, -28);
   copy_ie_register = transmit_dma_done;
   call enable_cpu_interrupts;
   call write_io_port(interrupt_enable_register,
                        transmit_dma_done);
   do while (copy_ie_register = transmit_dma_done);
   end;    /* loop until the interrupt handler
              takes care of the TDD interrupt -
              it sets IE_REG to 4 */
   call write_io_port (command_register, load_and_send);
   do while (mod(srf,2) = 0);
      call read_io_port (interrupt_status_reg, srf);
   end;  /* do while */
   call read_io_port (command_status_register, reg_value);


end transmit_packet;

/************************************************************/


HL_interrupt_handler: procedure external;



/* This routine is called from the low level
   8086 assembly language interrupt routine */

   DECLARE
```

```
     write_io_port entry (fixed bin (7), fixed bin (7)),
     read_io_port entry (fixed bin (7), fixed bin (7)),
     enable_cpu_interrupts          entry,
     disable_cpu_interrupts         entry,
     write_bar entry (pointer),
     match bit (1) static init ('1'b);

     /*  begin   */

  call disable_cpu_interrupts;
  call write_io_port(interrupt_enable_register,
                     disable_ni3012_interrupts);
  if (copy_ie_register = receive_block_available)
  then do;
     call write_bar (addr(receive_data_block));
     call write_io_port(high_byte_count_reg, 5);
    /* 1522 bytes */     call write_io_port(low_byte_count_reg, -14);

                         /* initiate receive DMA */

     call write_io_port(interrupt_enable_register,
                        receive_dma_done);
     copy_ie_register = receive_dma_done;
  end;   /* do */
  else
     if (copy_ie_register = receive_dma_done)
     then do;
        do i = 1 to 1500;
             if (transmit_data_block.data_bytes(i)
                                                 ~=
                 receive_data_block.data_bytes (i))
             then
                   match = 0;
         end;  /* iterative do */
         if ( match = 0) then
         do;
             put skip(2) edit ('*** warning ***')
                                (col(30),a);
             put skip edit ('*** Packet Error ***')
                            (col(25),a);
        end;  /* ift */
        operation = packet_received;
     end;
     else
        if (copy_ie_register = transmit_dma_done)
        then do;
             call write_io_port(interrupt_enable_register,
                                 receive_block_available);
             copy_ie_register = receive_block_available;
        end;   /* if then do */
end HL_interrupt_handler;
```

```
/***************************************************************/

command_status_codes: procedure (command_status)
                      external returns (char (30) varying);


   DECLARE

      command_status fixed bin (7);


   if command_status = 0 then
         return ('SUCCESS');
   else
   if command_status = 1 then
         return ('SUCCESS WITH RETRIES');
   else
   if command_status = 2 then
         return ('ILLEGAL COMMAND');
   else
   if command_status = 3 then
         return ('INAPPROPRIATE COMMAND');
   else
   if command_status = 4 then
         return ('FAILURE');
   else
   if command_status = 5 then
         return ('BUFFER SIZE EXCEEDED');
   else
   if command_status = 6 then
         return ('FRAME TOO SMALL');
   else
   if command_status = 8 then
         return ('EXCESSIVE COLLISIONS');
   else
   if command_status = 10 then
         return ('BUFFER ALIGNMENT ERROR');

end command_status_codes;

/***************************************************************/

diagnostic_codes: procedure (diag_status)
                  external returns (char(30) varying);

   DECLARE

      diag_status fixed bin (7);

   if diag_status = 0 then
      return ('SUCCESS');
   else
   if diag_status = 1 then
      return ('NM10 MICROPROCESSOR MEMORY ERROR');
   else
   if diag_status = 2 then
      return ('NM10 DMA ERROR');
   else
   if diag_status = 3 then
      return ('TRANSMITTER  ERROR');
   else
   if diag_status = 4 then
      return ('RECEIVER ERROR');
   else
   if diag_status = 5 then
      return ('LOOPBACK FAILURE');

end diagnostic_codes;

/*****************************************************************/



end;  /* procedure boardtest */
```

# LIST OF REFERENCES

1. Wasson, W.J., Detailed Design of the Kernel of a Real-Time Multiprocessor Operating System, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1980.

2. Reed, D.P. and Kanodia, R.J., "Synchronization with Eventcounts and Sequencers," Communication of the ACM, Volume 22, p. 115-123, February 1979.

3. Rapantzikos, D.K., Detailed Design of the Kernel of a Real-Time Multiprocessor Operating System, M.S. Thesis, Naval Postgraduate School, Monterey, California, March 1981.

4. Cox, E. R., A Real-Time, Distributed Operating System for a Multiple Computer System, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1981.

5. Klinefelter, S.G., Implementation of a Real-Time, Distributed Operating System for a Multiple Computer System, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1982.

6. Rowe, W. R., Adaptation of MCORTEX to the AEGIS Simulation Environment, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1984.

7. Dijkstra, E. W., "Cooperating Sequential Processes," Programming Languages, F. Genuys. Ed. Academic Press, New York, 1968.

8. Hoare, C.A.R., "Monitors: An Operating System Structuring Concept," Communications of the ACM, October 1974.

9. Xerox Corporation, The Ethernet - A Local Area Network : Data Link Layer and Physical Layer Specifications, Version 1.0, September 1980.

10. Swan, R. J., and others, CM* - A Modular Multi-Microprocessor, Proceedings of the National Computer Conference, 1977.

11. Green Paul E., Computer Network Architectures and Protocols, Plenum Press, New York, May 1983.

12. InterLAN Corporation, NI3010 MULTIBUS Ethernet Communications Controller User Manual, 1982.

13. Digital Research, CP/M-86 Programmer's Guide, 1981.

14. Digital Research, CP/M-86 System Guide, 1981.

15. Perry, M. L., Logic Design of a Shared Disk System in a Multi-Micro-Computer Environment, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1983.

16. INTEL Corporation, PL/M-86 Programming Manual for 8080/8085 Based Development Systems, 1980.

17. Digital Research, PL/I Language Reference Manual, 1982.

18. Digital Research, Programmer's Utilities Guide for the CP/M-86 Family of Operating Systems, 1982.

19. INTEL Corporation, ISIS-II User's Guide, 1978.

20. INTEL Corporation, MCS-86 Assembler Operating Instructions for ISIS-II Users, 1978.

21. INTEL Corporation, MCS-86 Macro Assembly Language Reference Manual, 1979.

INITIAL DISTRIBUTION LIST

| | | No. of copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22341 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943 | 2 |
| 3. | Department Chairman, Code 52<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 4. | Dr. M. L. Cotton, Code 62Cc<br>Department of Electrical Engineering<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 5. | Dr. Uno R. Kodres, Code 52Kr<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943 | 3 |
| 6. | LT David J. Brewer<br>7659 Cherrywood Dr.<br>Charleston Heights, South Carolina 29405 | 1 |
| 7. | Daniel Green, Code 20F<br>Naval Surface Weapons Center<br>Dahlgren, Virginia 22449 | 1 |
| 8. | CAPT J. Donegan, USN<br>PMS 400B5<br>Naval Sea Systems Command<br>Washington, D.C. 20362 | 1 |

9. RCA AEGIS Data Repository 1
RCA Corperation
Government Systems Division
Mail Stop 127-327
Moorestown, New Jersey 08057

10. Library (Code E33-05) 1
Naval Surface Warfare Center
Dahlgren, Virginia 22449

11. Dr. M. J. Gralia 1
Applied Physics Laboratory
Johns Hopkins Road
Laurel, Maryland 20707

12. Dana Small 1
Code 8242, NOSC
San Diego, California 92152

13. Loris O. Brewer 1
Route 2, Box 152
Fair Play, Missouri 65649

14. Jon Wilson 1
Case Rixon Communications
2120 Industrial Parkway
Silver Spring, Maryland 20904

15. Lubna Ejaz 1
Systems Analyst
Suburban Bank
6495 New Hampshire Avenue
Hyattsville, Maryland 20783

16. Kevin Acoveno 1
Case Rixon Communications
2120 Industrial Parkway
Silver Spring, Maryland 20904

17. C.H. Hudson 1
Route 4, Box 224
Stockton, Missouri 65785

18. Philip D. Sevold 1
6413 Lomack Court
Las Vegas, Nevada 89128

19. Karen Badner 1
11524 Colt Terrace
Silver Spring, Maryland 20902

| | | |
|---|---|---|
| 20. | Warren Chin-Lee<br>Sr. Systems Analyst<br>Suburban Bank<br>6495 New Hampshire Avenue<br>Hyattsville, Maryland 20783 | 1 |
| 21. | Jorge Herrera<br>Systems Programmer<br>Suburban Bank<br>6495 New Hampshire Avenue<br>Hyattsville, Maryland 20783 | 1 |
| 22. | Richard Zierdt<br>4707 Coachway Drive<br>Rockville, Maryland 20852 | 1 |

211147

Thesis
B80483 Brewer
c.1 A real-time executive
for multiple-computer
clusters.